

This Page Is Inserted by IFW Operations
and is not a part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

**As rescanning documents *will not* correct images,
please do not report the images to the
Image Problem Mailbox.**



INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification: G06F 3/06	A1	(11) International Publication Number: WO 00/49488 (43) International Publication Date: 24 August 2000 (24.08.2000)
(21) International Application Number: PCT/GB00/00550 (22) International Filing Date: 17 February 2000 (17.02.2000) (30) Priority Data: 9903490.2 17 February 1999 (17.02.1999) GB (60) Parent Application or Grant MEMORY CORPORATION PLC [/]; (). SINCLAIR, Alan, Welsh [/]; (). OUSPENSKAIA, Natalia Victorovna [/]; (). TAYLOR, Richard, Michael [/]; (). GOROBETS, Sergey, Anatolievich [/]; (). SINCLAIR, Alan, Welsh [/]; (). OUSPENSKAIA, Natalia Victorovna [/]; (). TAYLOR, Richard, Michael [/]; (). GOROBETS, Sergey, Anatolievich [/]; (). MCCALLUM, William, Potter ; ().	Published /	
(54) Title: MEMORY SYSTEM (54) Titre: SYSTEME DE MEMOIRE (57) Abstract A memory system (10) having a solid state memory (6) comprising non-volatile individually addressable memory sectors (1) arranged in erasable blocks, and a controller (8) for writing to reading from the sectors, and for sorting the blocks into "erased" and "not erased" blocks. The controller performs logical to physical address translation, and includes a Write Pointer (WP) for pointing to the physical sector address to which data is to be written from a host processor. A Sector Allocation Table (SAT) of logical addresses with respective physical addresses is stored in the memory, and the controller updates the SAT less frequently than sectors are written to with data from the host processor. The memory may be in a single chip, or in a plurality of chips. A novel system for arranging data in the individual sectors (1) is also claimed. (57) Abrégé La présente invention concerne un système de mémoire (10) comportant une mémoire électronique (6) contenant des secteurs de mémoires (1) non volatiles adressables un à un placés dans des blocs effaçables, et un contrôleur (8) assurant les opérations d'écriture et de lecture dans les secteurs, et assurant le classement des blocs en blocs "effacés" et "non effacés". En outre, ce contrôleur assure la conversion d'adresse logique en adresse physique, et applique un pointeur d'écriture (WP) désignant l'adresse secteur physique où les données provenant du processeur hôte doivent être écrites. La mémoire conserve une table d'affectation de secteur (SAT) donnant pour chaque adresse logique une adresse physique correspondante, le contrôleur mettant à jour les SAT moins fréquemment qu'il ne fait d'écriture secteur avec les données provenant du processeur hôte. La mémoire peut être réalisée sous la forme d'un seul microcircuit ou de plusieurs microcircuits. Par ailleurs, cette invention concerne un nouveau système d'organisation des données dans les secteurs (1).		

PCT

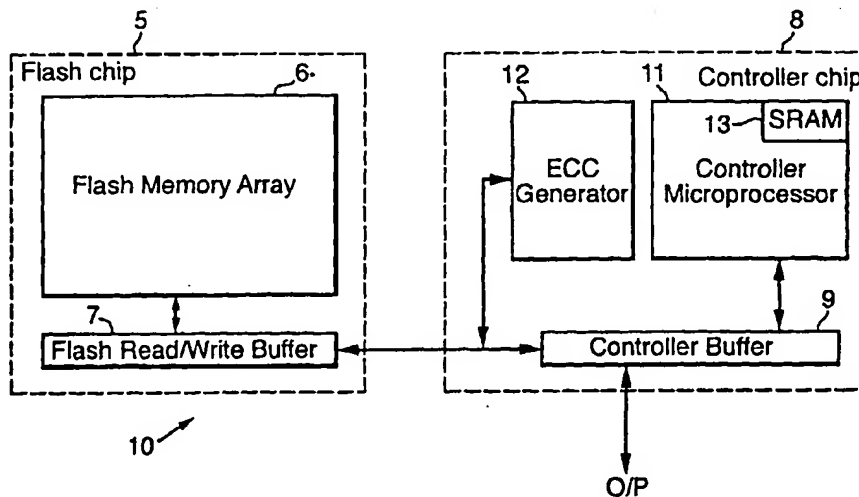
WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau



INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 : G06F 3/06		A1	(11) International Publication Number: WO 00/49488
			(43) International Publication Date: 24 August 2000 (24.08.00)
(21) International Application Number: PCT/GB00/00550		(81) Designated States: GB, JP, KR, SG, US, European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).	
(22) International Filing Date: 17 February 2000 (17.02.00)			
(30) Priority Data: 9903490.2 17 February 1999 (17.02.99) GB		Published With international search report.	
(71) Applicant (for all designated States except US): MEMORY CORPORATION PLC [GB/GB]; The Computer House, Dalkeith Palace, Dalkeith, Edinburgh EH22 2NA (GB).			
(72) Inventors; and			
(75) Inventors/Applicants (for US only): SINCLAIR, Alan, Welsh [GB/US]; 14059 Mango Drive #D, Del Mar, CA 92014 (US). OUSPENSKAIA, Natalia Victorovna [RU/RU]; 1-Murinski Prospect 29/20 apart. 63, St. Petersburg, 194100 (RU). TAYLOR, Richard, Michael [GB/GB]; Old Sawmill House, 41 Newmills Road, Dalkeith, Midlothian EH22 2AQ (GB). GOROBETS, Sergey, Anatolievich [RU/GB]; 1FI, 16 East Mayfield, Edinburgh EH9 1SE (GB).			
(74) Agents: MCCALLUM, William, Potter et al.; Cruikshank & Fairweather, 19 Royal Exchange Square, Glasgow G1 3AE (GB).			

(54) Title: MEMORY SYSTEM



(57) Abstract

A memory system (10) having a solid state memory (6) comprising non-volatile individually addressable memory sectors (1) arranged in erasable blocks, and a controller (8) for writing to reading from the sectors, and for sorting the blocks into "erased" and "not erased" blocks. The controller performs logical to physical address translation, and includes a Write Pointer (WP) for pointing to the physical sector address to which data is to be written from a host processor. A Sector Allocation Table (SAT) of logical addresses with respective physical addresses is stored in the memory, and the controller updates the SAT less frequently than sectors are written to with data from the host processor. The memory may be in a single chip, or in a plurality of chips. A novel system for arranging data in the individual sectors (1) is also claimed.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece	ML	Mali	TR	Turkey
BG	Bulgaria	HU	Hungary	MN	Mongolia	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MR	Mauritania	UA	Ukraine
BR	Brazil	IL	Israel	MW	Malawi	UG	Uganda
BY	Belarus	IS	Iceland	MX	Mexico	US	United States of America
CA	Canada	IT	Italy	NE	Niger	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NL	Netherlands	VN	Viet Nam
CG	Congo	KE	Kenya	NO	Norway	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NZ	New Zealand	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	PL	Poland		
CM	Cameroon	KR	Republic of Korea	PT	Portugal		
CN	China	KZ	Kazakhstan	RO	Romania		
CU	Cuba	LC	Saint Lucia	RU	Russian Federation		
CZ	Czech Republic	LJ	Liechtenstein	SD	Sudan		
DE	Germany	LK	Sri Lanka	SE	Sweden		
DK	Denmark	LR	Liberia	SG	Singapore		
EE	Estonia						

Description

5

10

15

20

25

30

35

40

45

50

55

MEMORY SYSTEM

5 The present invention relates to a solid state memory system for data storage and retrieval, and to a memory controller for controlling access to a non-volatile memory of a solid state
10 memory system. In particular, the invention relates to FLASH memory systems and controllers for FLASH memories.

15 FLASH EEPROM (electrically erasable programmable read only memory) devices are commonly used in the electronics industry for non-volatile data storage. Various types of FLASH memory
20 devices exist, including devices based on NAND type memory cells, AND type memory cells, or NOR type memory cells. Such devices may have different types of interfaces to the host processor system(s) for which they are designed to interface,
25 for example they may use a serial access type interface (as commonly used in many NAND and AND type devices) or a random access type interface (as used in some NOR type devices). The present invention is intended to be applicable, in appropriate
30 forms, to at least some and preferably all of these different types of memory devices.

35 It is known to use solid state memory systems to try to emulate magnetic disc storage devices in computer systems. It is an aim of the industry to try to increase the speed of
25 operation of solid state memory systems so as to better emulate magnetic disc storage.

40 According to a first aspect of the present invention we provide a memory system for connection to a host processor,
45 the system comprising:
a solid state memory having non-volatile memory sectors which are individually addressable and which are arranged in erasable blocks of sectors, each said sector having a physical
50 address defining its physical position in the memory;

and a controller for writing data structures to and reading data structures from the memory, and for sorting the blocks of sectors into blocks which are treated as erased and blocks which are treated as not erased; wherein the controller

includes:

means for translating logical addresses received from the host processor to physical addresses of said memory sectors in the memory;

a write pointer (hereinafter referred to as the Write Pointer (WP)) for pointing to the physical address of a sector to which data is to be written to from the host processor, said Write Pointer (WP) being controlled by the controller to move in a predetermined order through the physical addresses of the memory sectors of any block which is treated as erased and, when the block has been filled, to move to another of the erased blocks;

wherein the controller is configured so that, when a sector write command is received from the host processor, the controller translates a logical address received from the host processor to a physical address to which data is written by allocating for said logical address that physical address to which said Write Pointer (WP) is currently pointing; and wherein the controller is configured to compile a table of logical addresses with respective physical addresses which have been allocated therefor by the controller (this table being hereinafter referred to as the Sector Allocation Table or SAT), and wherein the controller updates the SAT less frequently than memory sectors are written to with data from the host processor.

By not updating the SAT every time data from the host processor is written to a sector in the memory, but instead updating the SAT on a less frequent basis, the present invention thus provides very high speed operation of solid

state memory, for example FLASH memory, thereby enabling good emulation of magnetic disk memory.

The physical sector addresses in the SAT are preferably ordered by logical sector address, whereby the Nth SAT entry contains the physical address of a sector to which data having logical address N has been written. When a sector read command is received from the host processor, the controller may look up a logical sector address received from the host processor in the SAT in order to obtain the physical sector address which the controller previously allocated to said logical sector address. The SAT is preferably stored in one or more of said blocks of memory sectors in the solid state memory, each block which contains any portion of the SAT hereinafter being referred to as a SAT block. Preferably the SAT is updated by rewriting one or more blocks of the SAT. By updating a whole block of SAT sectors at a time this significantly speeds up operation of the memory system.

There may be provided at least one block of sectors (hereinafter referred to as the Additional SAT Block (ASB)), containing modified versions of individual sectors of a said SAT block. Each sector in a said ASB block preferably contains the physical address of the sector of the SAT block which it updates, and the modified version of the said SAT sector. The purpose of an ASB is to cache individually in solid state memory modified sectors of the SAT so as to reduce the number of SAT block rewrites. When all the sectors in a said ASB block are written to with modified versions of SAT sector(s), the respective SAT block is rewritten so as to include all the modified versions in the ASB block and the ASB block is erased.

It will be appreciated that in the memory system of the present invention the physical address which is allocated to

5 any given logical address received from the host processor is
not dependent on the logical address itself. The controller
merely allocates the physical sector address to which the
10 Write Pointer is currently pointing.

5

As described above, the controller fills one said block which
is treated as erased before moving the Write Pointer (WP) on
15 to another block. The controller may conveniently be
configured to move the Write Pointer (WP) in a predetermined
10 order through the blocks which are treated as erased.

20 The controller may conveniently control the Write Pointer (WP)
so as to move sequentially, in ascending numerical order of
physical address, through the erased blocks, as each block is
15 filled with data written thereto. The control of the Write
Pointer (WP) may be cyclic in the sense that once the sectors
in the highest block, according to physical address order,
have been filled with data the WP is controlled by the
30 controller to wrap around to the block of sectors having the
20 numerically lowest physical addresses out of all the blocks
currently being treated by the controller as erased.

35 The controller may, alternatively, use another predetermined
order for writing data to the memory sectors. For example, the
25 controller may control the Write Pointer (WP) to move
sequentially in descending numerical order, according to
40 physical address, through the blocks which are treated as
erased. Another possibility would be to move in non-sequential
order through the physical sector addresses. For example, the
45 WP may move in descending numerical address order through the
physical sector addresses in each block which is treated as
erased, and move from block to block in some predetermined
order such as, for example, in ascending numerical order
50 according to the physical address of the first sector in each
35 said block.

5 It will be appreciated that many other predetermined orders
are possible for writing data to the sectors in the blocks
which are treated as erased. Furthermore, the controller could
10 use the erased blocks in any other order which need not be
predetermined, or which may be only partially predetermined.
Although generally not preferred, the erased blocks could even
15 be used in a random order.

10 The memory sectors in each said block of sectors are
preferably erasable together as a unit. The sectors may also
20 be individually erasable (for example where the solid state
memory is AND type memory). The controller is preferably
configured to control erase operations on the memory so as to
25 only erase whole blocks of memory sectors. A block of sectors
will be treated by the controller as an erased block if all
the memory sectors therein are erased sectors. If a block
contains one or more bad (i.e. defective) sectors, the
30 controller may define the whole block as being bad and treat
that block as a not erased block, whereby no data will be
written thereto. Alternatively, if a block contains one or
more bad sectors the controller may treat that block as an
35 erased block whereby the controller may still use good sectors
in the block to store data. In the latter case, though, the
25 memory system preferably includes a table identifying bad
sectors and the controller is configured to check whether the
40 next sector address to which the Write Pointer (WP) is to be
moved is the address of a bad sector and, if it is the address
of a bad sector, to control the Write Pointer to skip this bad
45 sector and move to the next sector address according to the
predetermined order in which the sectors are to be written to.

50 For the avoidance of doubt, any block which contains any good
(i.e. not defective) sectors which have already been written
35 to will be treated by the controller as a not erased block.

5 Furthermore, it is intended that the term "erased" sector covers not only a sector which has been erased, but also covers a sector which has never yet been written to, and so has not yet ever been erased. Thus, a block of sectors which
10 have never yet been written to is treated by the controller as an erased block.

15 Each block of sectors preferably has a physical block address defining its physical position in the memory. The physical
20 address of each said memory sector will preferably include the physical block address of the block in which it is located. The controller may advantageously be configured to compile a list of the physical block addresses of at least some of the blocks of sectors being treated as erased, which may be used
25 by the controller in order to quickly identify the next block of sectors to be written to. This list of addresses of erased blocks is preferably stored by the controller in a temporary memory which may be provided in the memory system, which
30 temporary memory may conveniently be an SRAM in a microprocessor of the controller, and may be created from information already stored in the solid state memory by the controller identifying the erased state of each block of
35 sectors. (This information will preferably be held in the form of a bitmap in the solid state memory, in which each block is recorded as an erased block or a not erased block.)

40 The controller is conveniently configured so that, when a sector write command is received by the controller from the host processor which command renders obsolete data previously
45 written to another sector, the controller stores in a temporary memory the address of the sector containing the now obsolete data. This temporary memory may conveniently be SRAM or DRAM provided in a microprocessor of the controller. If a
50 sector delete command, generated by a user, is received from the host processor by the controller, the controller
55

5 preferably marks as obsolete the sector to be deleted (without
physically erasing the sector). The controller may allow only
one block at any time, hereinafter referred to as the Current
10 Obsolete Block (COB), to contain one or more sectors
5 containing obsolete data which was written by the Write
Pointer (WP), and when all the sectors in the COB contain
obsolete data, the COB is immediately erased. This is a
15 particularly suitable scheme for the case where the Write
Pointer (WP) moves sequentially through the memory sector
10 addresses in each block which is treated as erased before
moving on to the next block. In such a scheme, a series of
20 obsolete sectors to be deleted (which may, for example,
contain part of a user data file which has been rewritten)
will in most cases all be in the same block. When a series of
25 15 sectors are rewritten in a different order to that in which
they were previously written, this may create obsolete sectors
in more than one block. Where a sector in a block other than
the COB is to contain obsolete data, the controller preferably
30 relocates any data in valid (not obsolete) sectors in the COB
20 to another block, which may be the block to which the Write
Pointer (WP) is currently pointing, and then erases the COB.
Said sector in the block other than the COB is then marked as
35 obsolete and this other block is now the COB. Rather than
writing the relocated data to the current location of the
25 Write Pointer, the memory system may include a second write
pointer, hereinafter referred to as the Relocation Pointer
40 (RP), for pointing to the physical address of the sector to
which such relocated data is to be written, the Relocation
Pointer (RP) always being in a different block of sectors to
45 30 the Write Pointer (WP). This has the advantage of preventing
relocated data from being intermingled with data structures
directly ordered to be written by the host processor i.e.
50 written by the Write Pointer (WP).

5 Generally, only two types of data are written to the solid
state memory from the host processor. These are file data and
system data. To further reduce the number of reallocations and
10 erasures, the memory system may further include a third write
5 pointer, hereinafter referred to as the System Write Pointer
(SWP), which points to the physical address of the sector to
which system data is to be written from the host, the SWP
15 always being in a different block to the Write Pointer (WP)
(and in a different block to the Relocation Pointer, if there
10 is one). System data will preferably be identified during
initialisation of the system and will be updated as necessary
20 during operation.

Where both a write pointer (WP) and a system write pointer
25 15 (SWP) are provided, file data will in this case always be
written to the addresses pointed to by the Write Pointer (WP).
Both the Relocation Pointer (RP) and System Write Pointer
(SWP) are preferably controlled to move through the physical
30 addresses of the memory sectors in said blocks which are
20 treated as erased in a similar manner to the Write Pointer
(WP). Thus, when all the (good) sectors in a said block have
been filled with relocated data or system data, the respective
35 one of the Relocation Pointer (RP) and the System Write
Pointer (SWP) moves on to the next address defined by the
25 controller to be used from the physical addresses of all the
40 sectors in the blocks treated as erased.

Where a System Write Pointer (SWP) is provided, the controller
will preferably allow at least two blocks which contain one or
45 30 more obsolete sectors to exist at any time, one being said COB
and the other being a Current Obsolete System Block (COSB)
containing one or more obsolete system data sectors. If any
50 system data sectors need to be relocated in order to allow the
COSB to be erased, the relocated system data is preferably

5 sent to the address to which the System Write Pointer (SWP) is currently pointing.

10 In fact, there may temporarily exist more than two blocks (the
5 COB and COSB) containing obsolete data at any one time. It is possible that when the COB, for example, needs to be erased (obsolete data has just been created in another block) one of
15 the write pointers may be pointing thereto i.e. the WP is still writing to the block which is currently the COB. Where
10 this is the case the controller preferably proceeds with creating the new COB but postpones the erasure of the old COB
20 (which is hereinafter treated as the Pending Obsolete Block (POB)) until all erased sectors in the POB have been filled and the write pointer moves on to the next erased block to be
25 used, as defined by the controller. At this time any valid (not obsolete) data in the POB is relocated and the POB is erased.

30 In addition to writing data structures to the memory from the
20 host processor, the controller may also generate and write to the memory data designated as control information. The controller preferably writes such control information in
35 separate ones of the blocks of memory sectors to those in which data structures received from the host processor are
25 written. Blocks for storing such control information, hereinafter referred to as Control Blocks (CBs), will be
40 updated periodically by the controller and will be accessed during initialisation, and occasionally during operation, of the memory system.

45 30 The controller preferably stores in a temporary memory (which may be a RAM provided in the memory system or which may
50 conveniently be an embedded SRAM or DRAM in a microprocessor of the controller) a list of logical sector addresses for data
35 structures which have been written by the Write Pointer (WP)

55

5 since the SAT was last updated. This list stored in the SRAM
is hereinafter referred to as the ~~Write Sector List (WSL)~~. The
logical addresses in the WSL are advantageously stored in the
10 order in which they were written to the non-volatile sectors
5 in the memory. Conveniently, for a group of consecutively
written sectors, the WSL entry may therefore be written as the
first sector logical address and the sector group length i.e.
15 the number of sectors written. Each said sector group is
defined so as not to span more than one block of sectors.

10

The controller advantageously also stores in said temporary
20 memory the order in which blocks have been used by the Write
Pointer (WP) for writing data since the last update of the
SAT. This is stored in the form of a list of block addresses
25 of the blocks in which the updated sectors whose addresses are
held in the WSL are located. This list of block addresses is
hereinafter referred to as the Write Block List (WBL). It will
be appreciated that since the memory system, by virtue of the
30 WSL and WBL, contains knowledge of the location in physical
20 memory which was allocated for the first logical address in
said group of consecutively written sectors, the controller
can thus always access the correct physical sector for each
35 logical sector address in a said group of consecutively
written sectors written since the last SAT update, using the
25 WSL and WBL. The WSL will preferably have a predetermined size
and once the WSL is full one or more SAT blocks (and/or ASBs)
40 may be updated and the WSL and WBL are emptied.

Preferably, the starting physical sector address, and the
45 30 links between blocks containing sectors to which data has been
written by the controller since the last SAT update, are also
stored in a Control Block of the solid state memory. By
50 storing the logical sector address for the user data stored in
each sector in the sector itself, for example in a header
35 field provided in the sector, the WSL and WBL can therefore

5 easily be recreated following any removal and restoration of
power to the system by scanning through the solid state
memory, reading the logical addresses in the sectors written
10 to since the last update of the SAT, until reaching a block
5 which is not full. This is the block which contained the Write
Pointer (WP) before removal or loss of power. This provides
high data security in the event of unexpected power removal
15 from the memory system.

10 Where a Relocation Pointer and a System Write Pointer are
included in the memory system, the controller preferably also
20 stores in said temporary memory (e.g. SRAM or DRAM in the
controller microprocessor) similar lists of logical sector
addresses corresponding to sectors in the memory to which
25 relocated data or system data has been written to
respectively, which lists are hereinafter referred to as the
Relocation Sector List (RSL) and Write System Sector List
(WSSL) respectively. The controller may also store in said
30 temporary memory corresponding lists of the order of blocks
20 which have been used by the RP and the SWP, similar to the
Write Block List, and these two lists will hereinafter be
referred to as the Relocation Block List (RBL) and the Write
35 System Block List (WSBL). Moreover, the starting physical
sector address, and the links between blocks containing
25 sectors to which relocated data or system data has been
written since the last SAT update may also be stored in at
40 least one said Control Block (CBs) of the solid state memory
whereby the RSL and WSSL can be recreated following any
removal and restoration of power to the host processor by
45 30 simply scanning the memory and reading the logical addresses
in the sectors written to by the RP and SWP respectively,
since the last update of the SAT.

50 Each said sector in any of the above-described embodiments may
35 consist of a single "page" of memory i.e. one row of memory

5 cells in a said block of memory sectors. However the invention
is not limited exclusively to such a sector format and in some
cases (for example when using random access NOR type memory)
each said sector may be less than, or greater than, one page.
10 Moreover, in the latter case not all said sectors need
necessarily be of the same size. For example, a data
organisation scheme such as that described in our earlier
15 International Patent Application No. PCT/GB99/00188 could be
used by the controller to form sectors of appropriate sizes so
10 as to avoid individual defects (of sub-sector size) which may
be present in the solid state memory.

20 Each sector is, as aforesaid, individually addressable. Each
sector may comprise a plurality of sector portions which are
25 also each individually addressable and the controller may
write to, and read from, each sector portion individually. It
will be appreciated that the smallest possible sector portion
size is the minimum addressable unit of the memory. In NOR
30 type memory, for example, the minimum addressable unit of
20 memory is commonly 1 byte.

35 The controller preferably writes data to, and reads data from,
the memory sectors in uniformly sized data segments. Where all
the memory sectors are the same size, each said data segment
25 is preferably equal in size to the size of a said memory
sector. Each data segment may comprise data structures from
40 the host processor (e.g. file or system data) and/or data
generated by the controller.

45 30 Where the solid state memory is based on NAND type devices,
the controller preferably stores in said one or more Control
Blocks a list of the block addresses of blocks in the non-
volatile memory containing bad sectors (hereinafter referred
50 to as the Bad Block List (BBL)), and the controller treats
35 each such block as a "not erased" block, so that it will not

5 appear in the list of erased blocks which may be stored in temporary memory, and the controller will not write any data to that block.

10 5 Where the memory is based on AND type devices, the controller preferably stores in said one or more Control Blocks (CBs) a list of addresses of any bad sectors, and the controller
15 controls the said write pointer(s) to use the good sectors in any block containing at least one bad sector, and to skip any
20 bad sectors. It will be appreciated that in the latter case where a block containing one or more bad sectors is to be
erased the good (i.e. non-defective) sectors in the block are
erased individually during a block erase operation.

25 15 The controller advantageously also stores in said one or more Control Blocks a list of the block addresses of all SAT
blocks. This list is preferably in the form of a plurality of
list portions, each said portion being hereinafter referred to
30 as a Table Block List (TBL), and each said portion containing
20 the block addresses of a group of logically contiguous SAT
blocks and any corresponding ASBs.

35 The controller preferably stores the block addresses of said one or more Control Blocks in a dedicated block of the memory
25 hereinafter referred to as the Boot Block (BB). Other
40 important information required for data security may also be stored in the Boot Block, for example the list of bad blocks
(or bad sectors). Preferably, the first block of sectors in
the memory which does not contain any bad sectors is
45 30 designated as the Boot Block (BB).

50 Preferably, the controller will only use blocks containing all good sectors as SAT blocks, Control Blocks, ASBs or BBs.

5 A cache may be provided in temporary memory (for example RAM
in the memory system, such as SRAM or DRAM in the controller
microprocessor), in which the controller stores a group of
10 contiguous SAT entries including the SAT entry most recently
5 accessed from the SAT (by the controller). This further
improves address translation speed. Further increase in speed
of address translation may be achieved by creating in said
15 temporary memory a list of physical addresses of all ASBs and
the SAT blocks with which they are associated (hereinafter
10 referred to as the ASB List or ASBL) which is updated each
time a SAT sector write operation is performed. Similarly, the
20 positions of the TBLs in the Control Block(s) may also be
stored in said temporary memory so as to allow even faster
logical-to-physical sector address translation.

25 15
The solid state memory may comprise a single memory array in
the form of a single memory chip, or may comprise a plurality
of memory arrays in the form of a plurality of memory chips.
30 Where the memory comprises a plurality of chips, the
20 controller advantageously forms the memory sectors in the
plurality of memory chips into a multiplicity of virtual
blocks, each said virtual block comprising one erasable block
35 of memory sectors from each said memory chip, and the
controller preferably sorts said virtual blocks into ones
25 which are treated as erased and ones which are treated as not
erased. The controller preferably compiles a list of the
40 virtual blocks treated as erased and stores this in temporary
memory in the memory system, which may be SRAM in a
microprocessor of the controller. The controller preferably
45 30 controls the Write Pointer (WP) (and the RP and SWP, where
provided) to move from one chip to another for each
consecutive sector write operation, starting at one sector in
one erasable block of the virtual block and moving
50 consecutively to one sector in each of the other erasable
35 blocks in the virtual block until one sector has been written

5 in each erasable block of the virtual block, and then moving
back to the chip in which the first sector was written and
proceeding in a similar manner to fill another one sector in
10 each erasable block of the virtual block, and so on until the
5 virtual block is full of data. The Write Pointer (WP) then
moves on to the next virtual block in said list of virtual
blocks being treated as erased, and fills this next virtual
15 block in a similar manner. The controller is preferably
configured so that for every n contiguous sector write
10 operations the controller executes, where n is less than or
equal to the number of solid state memory chips in the memory
20 system, the controller writes substantially concurrently to
one sector in each of n of the chips. The controller
preferably carries out erasure of any said virtual block by
15 concurrently erasing all the erasable blocks in the virtual
block.

It will be appreciated that the controller of the memory
30 system may be substantially implemented in circuitry as a
20 controller device, but will preferably be implemented, at
least in part, as firmware held in the memory of a controller
device. The controller may be integrally formed on the same
35 chip (or one of the same chips) as the solid state memory.

25 According to a second aspect of the invention we provide a
memory system for connection to a host processor, the memory
40 system comprising:
a solid state memory comprising a plurality of solid state
memory chips each having non-volatile memory sectors which are
45 30 individually addressable and which are arranged in erasable
blocks of sectors, each said sector having a physical address
defining its physical position in the memory;
and a controller for writing data structures to and reading
50 data structures from the memory, wherein:

5 the controller forms the erasable blocks into virtual blocks,
each said virtual block comprising an erasable block from each
of the memory chips, and the controller sorts the virtual
10 blocks into ones which are treated as erased and ones which
5 are treated as not erased, and the controller fills one
virtual block with data prior to moving on to the next virtual
block to be filled, and each virtual block is filled by
15 writing to the memory sectors thereof in a repeating sequence
in which the controller writes to one memory sector in each of
10 the erasable blocks of the virtual block one after another
whereby consecutively written sectors are in different chips.
20

Preferably, the controller is configured so that for every n
contiguous sector write operations the controller executes for
25 15 a multiple sector write command received from the host
processor, where n is less than or equal to the number of
solid state memory chips in the memory system, the controller
writes substantially concurrently to one sector in each of the
30 n of the chips.

20

According to a third aspect of the invention we provide a
controller for writing data structures to and reading data
35 structures from a solid state memory having non-volatile
memory sectors which are individually addressable and which
25 are arranged in erasable blocks of sectors, each said sector
having a physical address defining its physical position in
the memory, wherein the controller includes:
means for translating logical addresses received from a host
processor of a memory system in which the controller is used
45 30 to physical addresses of said memory sectors in the memory,
and for sorting the blocks of sectors into blocks which are
treated as erased and blocks which are treated as not erased;
and a Write Pointer (WP) for pointing to the physical address
50 of a sector to which is to be written to from the host
35 processor, said Write Pointer (WP) being controlled by the

5 controller to move in a predetermined order through the
physical addresses of the memory sectors in any block which is
treated as erased and, when the block has been filled, to move
10 to another of the erased blocks;

5 and wherein, when a sector write command is received by the
controller from the host processor, the controller translates
a logical sector address received from the host processor to a
15 physical address to which data is written by allocating for
said logical address that physical address to which said Write
10 Pointer (WP) is currently pointing;

and wherein the controller is configured to compile a table
20 (the SAT) of logical addresses with respective physical
addresses which have been allocated therefor by the
controller, and to update the SAT less frequently than memory
25 15 sectors are written to with data from the host processor.

According to a fourth aspect of the invention we provide a
method of controlling reading and writing of data structures
30 to and from a solid state memory having non-volatile memory
20 sectors which are individually addressable and which are
arranged in erasable blocks of sectors, each said sector
having a physical address defining its physical position in
35 the memory, the method comprising the steps of:

sorting the blocks of sectors into blocks which are treated as
25 erased and blocks which are treated as not erased;

40 providing a Write Pointer (WP) for pointing to the physical
address of a sector which is to be written to, and controlling
said at least one Write Pointer (WP) so as to move in a
predetermined order through the physical addresses of the
45 30 memory sectors of any block which is treated as erased and,
when the block has been filled, to move to another of the
erased blocks;

50 and, when a sector write command is received from the host
processor, translating a logical address received from the
35 host processor to a physical address to which data is written

5 by allocating for said logical address that physical address
to which said Write Pointer (WP) is currently pointing;
storing in non-volatile solid state memory a table (the SAT)
10 of logical addresses with respective physical addresses which
5 have been allocated therefor by the controller;
and updating the SAT less frequently than memory sectors are
written to with data from the host processor.

15 Preferred embodiments of the invention will now be described
10 by way of example only and with reference to the accompanying
drawings in which:

20 Fig.1 is a schematic illustration of one block of sectors in a
NAND type FLASH memory, showing three sectors therein;

25 Fig.2 is a block diagram of a memory system comprising a FLASH
15 chip and a controller chip;

Fig.3 is a schematic illustration of one page of data in a
NAND or AND type FLASH memory;

30 Fig.4 shows the structure of a Header field of the page of
Fig.3;

20 Fig.5 illustrates the format of a physical address (PA) of a
page;

Fig.6 illustrates a Control Block (CB) entry;

35 Fig.7 illustrates one entry in a Table Block List (TBL);

Fig.8 shows the format of a MAP entry;

25 Fig.9 shows the format of an entry in the ASB List (ASBL);

40 Fig.10 illustrates the format of a Current Obsolete Block
(COB) Structure;

Fig.11 is a table illustrating the order in which sectors are
written to in a virtual block of a multiple FLASH chip memory
45 system according to one embodiment of the invention;

Fig.12 shows the format of a Virtual Address (VA);

Fig.13 shows how the PA is obtained from the VA;

50 Fig.14 illustrates the timing of operations during a multiple
sector write to a multiple FLASH chip memory system according
35 to the invention;

5 Fig.15 is a block diagram of a controller chip;
Fig.16 is a table showing allocated memory capacity for a
memory system of the invention;
10 Fig.17 is a flow diagram showing an Address Translation
5 process;
Fig.18 is a flow diagram of the steps carried out at box 58 of
Fig.17;
15 Fig.19 is a block diagram of a multiple FLASH chip memory
system comprising four FLASH chips and a controller chip;
20 Fig.20 is a flow diagram of the steps carried out at box 56 of
Fig.17;
Fig.21 is a flow diagram of the steps carried out at box 44 of
Fig.17;
Fig.22 is a flow diagram of a sector read operation;
25 Fig.23 is a flow diagram of a sector write operation;
Fig.24 is a flow diagram of the steps carried out at box 161
of Fig.23;
Fig.25 is a flow diagram of the steps carried out at box 207
30 of Fig.24;
20 Fig.26 is a flow diagram of the steps carried out at box 160
227 of Fig.23;
Fig.27 is a flow diagram of a sector delete operation;
35 Fig.28 illustrates the physical partitioning of a page in NAND
or AND type FLASH memory;;
25 Fig.29 is an illustration of an alternative way of arranging
the data in the FLASH page of Fig.28;
40 Fig.30 is an illustration of a yet further way of arranging
the data in the FLASH page of Fig.28;
Figs.31(a) illustrates data in a buffer memory of the
45 30 controller prior to a sector write operation;

Fig.31(b) illustrates data in a FLASH page after completion of
50 a write operation, where the data is arranged according to the
embodiment of Fig.30;

Fig.32 is a table of controller commands used to transfer the data from the controller buffer to the FLASH memory during the write operation of Figs.31(a) and (b);

Fig.33 illustrates data in a buffer memory of the controller 5 after a read operation;

Fig.34 is a table of of controller commands used to transfer the data from the FLASH memory to the controller buffer during the read operation of Fig.33; and

Fig.35 is a schematic block diagram of an erasable block of 10 sectors in NOR type FLASH memory, showing three sectors therein.

Fig.1 illustrates schematically the physical page structure in 15 one block 4 of a FLASH memory array based on NAND type memory cells. Fig. 1 shows three pages 1, 2, 3 in the block 4. The page 1 in physical terms comprises one row of memory cells in a block of memory, the memory being partitioned into many such 30 blocks each comprising a multiplicity of rows of memory cells (i.e. a multiplicity of pages). Each page 1, 2, 3 is treated as one sector of physical memory space in the FLASH memory system which will be described and is 528 Bytes wide. Each 35 page 1 in the memory is individually addressable (for read/write and delete operations), and the pages are erasable 25 in blocks. We will now describe a memory system incorporating such a memory array. We will later additionally describe 40 memory systems based on AND or NOR type FLASH memory.

Fig.2 shows a memory system 10 incorporating a FLASH memory 45 chip 5 and a controller chip 8. The FLASH memory chip 5 comprises a FLASH memory array 6 and a read/write buffer 7 interfaced to a controller buffer 9 in the controller chip 8. The controller chip 8 further includes a controller 50 microprocessor 11 and an Error Correction Code (ECC) generator 35 and checker 12. The controller buffer 9 interfaces to a host

5 computer processor (not shown) connected to the memory system
10 via an output O/P of the controller chip 8. The controller
chip 8 (hereinafter referred to as the "controller"), controls
the reading and writing of data structures to and from the
10 memory array 6. The host processor 2 connected to the memory
system 10 sends read and write commands to the controller 8.
Data can be accessed by the host in 512 Byte portions or "host
15 data sectors", each of which has a logical sector address
(LA). The controller 8 receives an LA from the host processor
10 and translates this to a physical address as will be described
hereinbelow. In the present case (NAND type memory), each
20 physical address (PA) defines the physical position of a page
1 of the FLASH memory in the array 6. Each LA is in the form
of one 24 bit field. Accessing a PA using an LA is referred
25 to as address translation and is commonly the most frequent
operation needed on every read/write access. The controller 8
writes data to the memory array 6 in data segments, each
segment being 528 Bytes wide. For each 512 Bytes of data
30 received from the host (e.g. user file or system data) the
controller generates 16 Bytes of data comprising a 4 Byte
Header generated by the microprocessor 11 and a 12 Byte ECC
produced by the ECC generator and checker 12. The controller
35 organises this into a 528 Byte data segment which is written
to one page of the memory array 6, via the FLASH buffer 7.
25 The logical address (LA) of a host data sector is stored in
the 4 Byte Header in the FLASH Sector 1 in which that host
40 data sector is written. On a read operation the data stored in
the relevant sector of the FLASH memory array is read from
array 6, via the FLASH read/write buffer 7, into the
45 controller buffer 9 (and concurrently to the ECC generator and
checker to check for errors in the data), and the controller
reads the 4-byte header to check that the stored LA matches
50 the LA requested by the host computer, prior to allowing the
host computer to read the data from the controller buffer 9.

5 The Controller 8 manages the physical location of data written
to the memory 6 on an individual sector basis. As will be
later described in further detail, the controller stores in
10 the memory 6 a BitMap (MAP) of erased blocks and compiles in
5 SRAM in the microprocessor 11 a list (the Next Erased Block
(NEB) list) of at least some erased blocks, ordered in
ascending order of block physical addresses, which erased
15 blocks are to be used for writing to. The physical page
location at which a host data sector is written does not
10 depend on the logical address received from the host. Each
Host Data sector is written at an address defined by a cyclic
20 write pointer. Special write pointers are used for different
types of write operations; host file data writes are performed
at an address pointed to by the Data Write Pointer (WP), host
25 system data writes at an address pointed to by the System
Write Pointer (SWP). The Relocation Pointer (RP) is used for
writing sectors which were not directly ordered by a host.
Each of these write pointers has an identical behaviour: each
30 pointer moves sequentially through the pages of a block, then
20 moves to the first page of the next erased block in the Next
Erased Block (NEB) list. Blocks containing files which are
not erased are treated as "not erased" blocks and are skipped
35 when a pointer moves from one block to another (and are never
included in the NEB).

25

40 Sector Relocation Algorithm

When a block 4 of sectors is to be erased, so as to recover
sector space containing obsolete data, sectors must be
relocated from a block containing a combination of valid and
45 obsolete sectors to allow the block to be erased. In
principle, the controller 8 allows only one block
corresponding to a specific write pointer to contain obsolete
data sectors at any time. When a sector to be written by the
50 host would produce an obsolete sector in a second block, the

55

5 existing block must first be erased, after relocation of valid sectors, if necessary.

10 Therefore, numerous erasures and extensive relocations of
5 sectors are unavoidable when a majority of blocks are to
contain both valid and obsolete sectors. This occurs only
when a sequence of sectors written by a host as a part of a
15 file differs from the sequence in which they had previously
been written. This is not the normal case in most
10 applications. However, even in case of normal file write
operation, relocation of non-related data must be performed
20 from blocks containing a "head" and "tail" of a file. There
is a high probability that relocation of system data
intermingled with "other file" data will cause additional
15 erasure of another block and this will produce more
25 relocations from this block.

To reduce the total number of relocations and erasures system
30 data is therefore specifically identified and is always
20 written or relocated at the address of the System Write
Pointer (SWP). Information about system data is obtained
during initialisation process and is stored in the
35 microprocessor SRAM 13. It will be generally understood that a
data file is written to the FLASH memory by a file system in
25 the host computer processor. File data is partitioned into
40 clusters by the file system, where each cluster is a group of
contiguous host data sectors of (typically) 512Bytes. The file
system maintains tables and data structures relating to the
attributes of the files stored in the memory and the locations
45 30 of the clusters which form each file. These tables and
structures are stored in the FLASH memory (as system data) and
are updated by the file system whenever file data is written.
When file data is written to the memory it is accompanied by
50 system data (e.g. for Directory and File Allocation Tables)
35 which relate to the file. System data written in the memory

5 commonly includes BIOS Parameter configuration information,
one or two copies of the File Allocation Table (FAT) in which
each entry relates to a specific cluster, the Root Directory,
10 and Subdirectory information. The controller is configured to
5 recognise an operation to write a host system data sector,
thereby enabling it to treat this host data sector differently
to a host data sector of file data. A number of methods may be
15 used either singly or together to recognise system sector
writes, as follows:

- 10 1. System data is written with single sector write commands,
whilst file data may be written with multiple sector write
20 commands.
2. All sectors with LAs below the last sector address in the
file system Root Directory are system sectors. This address
25 can be determined from information held in a BIOS parameter
block stored in the memory by the host file system.
3. All sectors within Subdirectories are system sectors.
Subdirectory addresses and sizes can be identified by reading
30 all Root Directory and Subdirectory entries.
- 20 4. System sectors are often read by a file system immediately
before they are rewritten.

35 For the same purpose file data sectors to be relocated are
written at the address defined by a Relocation Pointer (RP),
25 and are therefore not intermingled with sectors written by the
40 host.

In a modified embodiment of the invention, an additional write
pointer may be provided to point to the location to which
45 30 relocated system data is to be written. This additional
pointer is referred to as the System Relocation Pointer (SRP),
and is always located in a different block to the WP and SP.

50 Block Erase Algorithm

5 No arbitrary selection of a block to be erased or scheduling
of background erasure is carried out in the present invention.
Listed erasure of a block containing obsolete sectors is
10 normally immediately performed when an obsolete sector in a
5 second block will result from a host sector write command
which is pending. Similarly, a block is immediately erased
when it contains totally obsolete control data structures as a
15 result of the rewriting of a control block. (Control blocks
are where the controller 8 writes certain control data, and
10 are described in further detail later).

20 Therefore, normally, since relocations cannot produce obsolete
data, there can exist not more than two blocks which contain
obsolete data; the Current Obsolete Block (COB) corresponding
25 15 to the Data Write Pointer (WP) and containing obsolete file
data and the Current Obsolete System Block (COSB)
corresponding to the System Write Pointer (SWP) and containing
obsolete system data. However, temporarily there may exist
30 one more obsolete block of each type. This will occur when a
20 block to be erased (obsolete data has just been created in
another block) at this moment also contains a write pointer of
any type. In this case erasure of such a block (designated
35 Pending Obsolete Block (POB)) has to be postponed until all
erased pages in this block have been used and the relevant
25 write pointer will be moved to another block. At this moment
40 the Pending Obsolete Block is immediately erased.

As aforementioned, erased block identity is maintained in a
BitMap (MAP) spanning the whole FLASH block address space, in
45 30 which MAP the erased state of each block is recorded. Erased
blocks are consumed for sector or control data writing
sequentially in block address order. No background erasure is
50 carried out. Any block containing one or more bad sectors is
treated as a Bad Block and is treated as a "not erased" block
35 by the controller.

5

Wear Levelling

10

15

20

The use of cyclic write pointers and single sector write management produces inherent wear levelling in the FLASH memory. However, the algorithm of erasing blocks as soon as they are populated with obsolete or deleted data produces a wear levelling characteristic which is a function of the sequences of sector write operations. If any further wear levelling is thought to be necessary, then separate additional techniques may be incorporated, such as occasional relocation of sectors in random blocks so as to allow these blocks to be erased.

Address Translation Principles

25

30

35

40

The principal address translation means is the Sector Address Table (SAT), which basically is a list of physical addresses of sectors, ordered by logical address. Thus the Nth SAT entry normally contains the physical address for sector with logical address N. The SAT is organised as a number of independent blocks (SAT blocks), and is updated by rewriting individual pages of the SAT blocks. A SAT block may have a dedicated Additional SAT Block (ASB) linked with it to allow modification of individual pages of the SAT Block. SAT pages are not rewritten after each sector write, but on a much less frequent basis, to minimise impact on sector write performance.

45

50

Therefore, the SAT will not contain the correct physical address for sectors written since the SAT was last updated. The logical addresses of such sectors are stored by the processor in its SRAM 13 in lists called the Write Sector List (WSL), Relocation Sector List (RSL) and Write System Sector List (WSSL). These lists exactly match the ordering of the sectors themselves which were written by a host or relocated from blocks before erasure. In the case of consecutively

55

5 written sectors the WSL and RSL entry defines first sector
logical address and the sector group length. The sector
groups cannot jump from one block to another. The
10 microprocessor 11 has knowledge of the starting point of the
5 sector series in Flash memory, and also the order in which
blocks were used for sector writing (special lists,
complimentary to those described above and created in the
15 processor SRAM, namely the Write Block List (WBL), Write
System Block List (WSBL) and Relocation Block List (RBL), are
10 used to store this information and are described in further
detail later), and so can calculate the physical location of
20 the sector.

The WSL, RSL and WSSL (and lists, complimentary to them: WBL,
25 15 WSBL and RBL) can be recreated by the microprocessor 11 after
removal and restoration of power to the memory system 10 by
reading the logical addresses in the headers of the sectors in
the series written since the last SAT rewrite. The starting
30 sector address in the series and the links between blocks
20 containing sectors in the series is obtained by the
microprocessor from entries in a special data structure called
the Control Block (CB) in Flash memory. (The Control Block
35 (CB) is described in detail later). This method gives high
security of data in the event of unexpected power removal from
25 the card.

40 It will be appreciated that where a System Relocation Pointer
(SRP) is included, as mentioned above, a System Relocation
Sector List (SRTL) and complementary System Relocation Block
45 30 List (SRBL) are also created and used in a similar manner as
described above with regard to the WSL, RSL and WSSL, and the
WBL, RBL and WSBL, respectively.

50 The organisation of data in a FLASH Sector (i.e. page) 1 to
35 which data is written by the controller 8, according to the

5 present embodiment, is illustrated in Fig. 3. The Sector 1
contains a first, 512 Byte information portion 1a which may,
for example, consist of a host data sector, followed by a 4
10 byte header portion 1b, followed in turn by 12 Bytes of ECC
5 1c. As shown in Fig. 4, the Header portion itself comprises a
Data Structure Type portion 20 and a Header Parameter 22 (e.g.
comprising the logical sector address (LA) of a host data
15 sector written in the information portion 1a. The Data
Structure Type can have values representing any one of the
10 following : Data Sector; Deleted Data Sector; Sector Address
Table (SAT) page; Additional SAT Block (ASB) page; Control
20 Block (CB) page; and Boot Block (BB) page.

Deleted Data Sector

25 A deleted data sector physically exists temporarily in the
FLASH memory only in blocks which are permitted to contain
obsolete or deleted sector data i.e. the COB or COSB. It is
identified by the "all zero" state of the Data Structure Type
30 field in the header, or other means as appropriate.

20

Sector Address Table (SAT)

35 The SAT is a series of entries containing the physical address
of logical sectors. Entry N contains the physical address for
logical sector N. Entries within a page occupy the 512 bytes
25 of the information portion 1a of the data stored in the page.
Because a SAT page is written in a single operation, the EEC
40 field may be used to protect the complete page, and separate
ECC fields for each entry in the SAT page are not required.

45 30 The SAT is actually stored as a series of blocks, each block
containing up to 64 SAT pages. A separate data structure, the
Table Block List (TBL), is held in the Control Block
50 (described later) to define the block addresses of all SAT
blocks. Each SAT entry defines the physical address of a
35 sector and occupies 3 bytes and as shown in Fig.5 comprises:

5
10
15
20
25
30
35
40
45
50
55

Chip Number	5 bits, allows 32 chips to be addressed
Block Number	13 bits, allows there to be 8192 blocks per chip
Sector Number	6 bits, allows up to 64 sectors per block.

The Header Parameter field on a SAT page data structure contains the SAT block and page number.

A Flash card with capacity 8MB and 8VB blocks can store approximately 16K sectors, and its SAT will therefore have approximately 16K entries. A 512-byte page within a SAT will contain 170 entries, and the SAT will therefore occupy almost 96 pages, which will occupy 6 blocks. A SAT for a large FLASH memory card (2GB, 8VB blocks) occupies 1543 blocks.

Additional SAT Block (ASB)

An Additional SAT Block (ASB) is a dedicated block which may be linked to a specific SAT block to allow single pages of the SAT block to be modified (i.e. rewritten). There can be several ASBs, each of which acts as an extension to the SAT block to which it is linked. When a SAT block is to be modified, it generally contains modified data in only a small number of its pages. The ASB allows only these modified pages to be rewritten. This is a much faster operation than the writing of every page and erasure of the obsolete block which is required for rewriting a SAT block. The Header Parameter portion of an ASB page contains the SAT block to which it is linked and the SAT page number within that block which it replaces. The format of the information portion 1a of data stored in an ASB page is identical to that of a SAT page.

Control Blocks (CBs)

The Controller 8 stores some control and address information in one or more Control Blocks (CBs). The CBs are updated

periodically by the controller 8 and must be accessed during initialisation of the memory system 10 or occasionally during operation. Information is stored in the CBs in entries of a fixed size which can be independently written. There are 9 entries per page in each CB. An entry relates to one of the following list of data types, which are identified by a CB Header field in the entry itself :

- Table Block List (TBL)
- Map of blocks with some fields in the entry corresponding to a file data write operation (WMAP)
- Map of blocks corresponding to a system data write operation (SMAP)
- Map of blocks corresponding to a relocate sector operation (RMAP)

When new data must be added to the CB, an additional entry of the appropriate type is added immediately following the last valid entry. For large cards the CB may occupy more than one block. The addresses of all CB blocks are stored in a Boot Block (BB) in the FLASH memory 6.

20

For purposes of data security the first page (designated Header Page) of each block of a CB (and the BB also) does not contain entries and has a full page format of like that of Fig. 3. The Header Parameter field of the Header 16 of this page consists of a Signature, which identifies the CB, and its Block Number (which is the block's serial number within the set of Control Blocks).

The Information Field 1a of the Header Page of the first CB block is occupied by a Block Link Info data. The Block Link Info data provides all necessary information to restore the WBL, WSBL and RBL if the system has to be initialised following a CB rewrite operation. It comprises Link fields collected from all MAPs (WMAPS, SMAPS and RMAPS) written since

5 the last SAT page write operation was performed and has to be
written to the Header Page of the first block of a new CB
during its rewrite operation. The Block Link Info is a timing
10 ordered list of 4 bytes entries, each of which contains a
5 block address of a block being visited by one of the write
pointers and a flag identifying which pointer it was. A
maximum number of blocks in memory space is allowed for CBs
15 and when this becomes full, active entries are rewritten to
the next available erased block(s) from the NEB (i.e. it is
10 compacted and corresponding Header Pages are added), and the
old CBs are erased. (At the same time a corresponding entry
20 has to be added to the Boot Block.) The information field of
a CB page contains 9 entries of equal length. The EEC field
of a CB page is not used or may be used for other purposes.

25 15
An entry in a CB is 56 Bytes wide and has the format shown in
Fig.6 and comprises a Header 24 identifying the data type of
the entry, an information field 26, and an EEC field 28. The
30 EEC is of the same form as is used for a full page.

20

Table Block List (TBL)

35 The CBs contain the Table Block List (TBL). The TBL contains
the addresses of a group of contiguous blocks of the SAT and
any corresponding ASBs. Multiple TBLs are normally required
25 to define all required SAT block addresses. The relevant TBL
is written immediately after a SAT block write operation or
40 allocation of a new ASB, to record the modified SAT or ASB
block locations. The TBL also contains fields which are used
to record write pointer positions at the moment when a SAT
45 30 page write operation is to be performed. Therefore, a TBL is
also written whenever a SAT page write operation is carried
out. One entry of the TBL is shown in Fig. 7 and occupies the
50 information field 26 of a CB entry. The first 1 Byte of the
TBL entry is No, a sequential number of the TBL entry. Each
35 TBL entry keeps values of 8 SAT - ASB block pairs, in other

5 words each TBL entry keeps values for SAT blocks from N to N+7
where N is No*8.

10 WP is the Write Pointer Page field, RP is the Relocation
5 Pointer field, and SWP is the System Write Pointer field.
These fields determine the position of the WP, RP and SWP
within the blocks after WSL, RSL, SSL release operation. WP,
15 RP and SWP are valid only when WSL, RSL or SSL release
operation is complete (i.e. at the last SAT write operation).
10 This condition is set with the Flag bit in the entry Header.
Flag=1 means the entry is the last one written during WSL or
20 RSL release and so WP field is valid.

3 reserved bytes are left for possible future additions.
15
25 SAT-ASB Pairs follow reserved fields, this is an array of 8
entries, each consists of SAT and ASB block addresses.

30 SAT_N is the number of the Nth SAT block and ASB_N is the number
20 of the ASB linked to the SAT_N . If SAT or ASB doesn't exist
the value of this field should be equal to zero.

35 New TBL entry should be added to CB each time SAT block is
relocated or new ASB is linked to SAT block.

25
40 MAP (WMAP, SMAP and RMAP)
The CBs contain various MAP entries. There are three
different types of MAP entry each of which corresponds to a
different type of write operation; WMAP - to a file data write
45 30 or delete operation, SMAP - to a system data write and RMAP -
to a sector relocation operation.

50 The information field of all the MAP entries has the same
format. It contains a BitMap defining the erase state of a
35 group of consecutive blocks. The erase state of 256 blocks is

5 defined by the 256 bits in a 32 byte field in the MAP. The
information field contains a Range field identifying the group
of blocks to which it relates. Another field defines the
10 destination block for a transition of the write pointer
5 between blocks. The MAP also contains fields identifying the
location of blocks in flash memory containing obsolete data;
ObsC is used for the COB (or COSB) and ObsP - for the Pending
15 Obsolete Block. If obsolete block is not present
corresponding field is set to 0. The EB field contains an
10 address of a block in which erasure is caused by current write
or delete sector operation. If there is no such block the EB
20 field is set to 0. This MAP entry format is illustrated in
Fig.8.

25 15 When one of the write pointers is moved from one block to
another, a corresponding MAP entry must be added to show the
use of an erased block (the BitMap field is updated) and to
record the link between the blocks (the Link field is
30 updated). When a write (or delete sector) operation produces
20 obsolete (or deleted) data in a new block, a corresponding MAP
also must be added to record a new obsolete block position
(ObsC or/and ObsP fields are updated), to indicate that a
35 block has to be erased (the EB field is updated) and to show
that a new erased block is going to appear (the BitMap field
25 is updated). Therefore, normally, at least two fields of a
MAP are written at the same time and this may be achieved in a
40 single page write operation.

Boot Block (BB)

45 30 The function of the Boot Block is to provide high security of
data in the event of unexpected power removal from the card
and at the same time avoid extensive scanning during
initialisation procedure. For large cards the BB contains
50 more than one block. The BB always occupies the first
35 nondefective block(s) in the card. For purpose of data

5 security there is a copy of the BB occupying the next
nondefective block(s) in the card. Both the BB and its copy
must be put to the same place after being rewritten.

10 5 The BB has the same structure as the Control Block and
contains the following types of entries :

- Signature
- 15 • Interleaving Enable.
- Bad Block List

20 • Control Block Pointers Table (CBPT)

20 When new data must be added to the BB, an additional entry of
the appropriate type is added immediately following the last
valid entry. The Signature and the BBL entries have just the
same format as described before (of course, the signature
25 field in the signature entry is different and unique). The
Control Block Pointers Table entry contains pointers to all
blocks of the CB and has to be updated immediately after the
CB is rewritten.

30 20 Previous Link (PL)

The CB(s) also contain the Previous Link (PL). The purpose of
35 the Previous Link is to provide all necessary information to
restore the WBL (described later) if the system has to be
initialised following a CB rewrite operation. It comprises
25 Link fields collected from all MAPs written since the last SAT
page write operation has been performed. The PL has to be
40 written only to a new CB during its rewrite operation.

Data Structures Stored in SRAM of Controller

45 30 Various data structures are stored in the SRAM 13 of
microprocessor including the following :

50 Write Sector List (WSL) (or "whistle")

5 The Write Sector List records the logical addresses of sectors
written after the last SAT write. Its purpose is to allow
correct logical-to-physical address translation for such
10 sectors. The WSL has capacity for 128 entries of length 4
5 bytes, where each entry stores the logical address of the
first sector in a group of consecutively written sectors and
the group length. The sector groups cannot jump from one
15 block to another. The WSL is empty immediately after a SAT
write.

10

20 The ordering of logical sector addresses in the WSL exactly
matches the order in which they were written. A non-volatile
copy of the WSL in Flash memory is therefore automatically
provided by the headers of the actual sectors written at
25 consecutive locations starting at that defined by the Write
Pointer (WP) at the time the SAT was last written. There is
therefore no need to explicitly make copies of the WSL to
Flash memory 6. If necessary, these sectors can be scanned
30 from a starting address defined in a Control Block field which
20 contains the position of the Write Pointer (WP) at the time
the SAT was last written, along with the Link fields from
subsequent MAP entries.

35

Search of the WSL is performed in reverse order, since only
25 the last entry for any logical sector is valid. Duplicate
earlier entries may not have any corresponding obsolete sector
40 located in Flash memory because sector relocations and block
erasures may have been performed. If preferred, the controller
is configured to simply remove duplicate entries from the WSL.

45

30

Two similar lists, the Relocation Sector List (RSL) and System
Sector List (SSL), are also compiled in the microprocessor
50 SRAM 13 recording the logical addresses of relocated sectors
and System data sectors (written to addresses pointed to by
35 the RP and SWP respectively) written since the last SAT write.

55

5 An ASB and/or a SAT block is supplemented with WSL, RSL or SSL
entries every time the WSL, RLS or SSL respectively is full.
10 This procedure is called WSL, RSL or SSL release. This
5 release can cause ASB release if necessary. ASB release
occurs when an ASB is full. When an ASB is full the
respective SAT block is rewritten and the ASB is erased.
15 Information about written pages in all ASBs should be stored
in RAM to avoid frequent ASB scanning. An ASB List (ASBL) for
10 this purpose is stored in the SRAM 13.

20 The ASBL is a list of all the ASB blocks currently linked with
SAT blocks, there being one entry in the ASBL for each ASB.
Fig.9 illustrates the format of one entry in the ASBL, where :
25 15 LWP = number of last written page in this ASB block
NVP = number of the valid pages in ASB block minus one.
ASB Page 0 ... ASB Page n = an array, index is ASB page
number, value is corresponding SAT page number.
30 N = pages per block.

20

Write Block List (WBL)

35 The Write Block List is complementary to the Write Sector List
and is created in microprocessor SRAM 13 to define the blocks
within which the sectors in the WSL are located. The WBL is
25 empty immediately after WSL release.

40 The WSL and WBL are recreated by a scanning process during
initialisation of the memory system. The lists in SRAM which
are created in this way will exactly match the lists which
45 30 existed before power was last removed.

Two similar lists to the WBL named the Relocation Block List
50 (RBL) and System Block List (SBL), are also compiled and
stored in the SRAM 13, these lists being complimentary to the
35 RSL and SSL respectively. The RBL and SBL define the blocks

55

within which the sectors in the RSL and SSL respectively are physically located, and are of similar format to the WBL. The RSL, RBL, SSL and SBL can also be recreated by a scanning process during initialisation of the memory system.

Current Obsolete Block (COB)

Only one block is allowed to exist which contains obsolete or deleted sector data written by the Write Pointer (WP). This is named the Current Obsolete Block (COB). When obsolete or deleted sector data is created in another block, an erase operation must be performed immediately on the block defined as the COB. The current block address of the COB and also a list of sectors which became obsolete or were deleted in this block are stored in the microprocessor RAM 13 as a data structure hereinafter referred to as the COB Structure. The COB Structure is established during initialisation by copying a field containing obsolete block address from the latest MAP entry, and then adding obsolete sector addresses after the reconstructed WSL and WBL have been analysed and reading deleted sector addresses from this block. The COB Structure is updated every time a new delete sector operation is performed or a new obsolete sector is created, and the block address is copied to a current MAP entry in the CB every time obsolete data is created in a new block.

25

There is also allowed to exist one block which contains obsolete or deleted sector data written by the SWP, named the Current Obsolete System Block (COSB). The COSB Structure is also stored in the SRAM 13 in a similar manner to the COB Structure. The COB and COSB each have the format shown in Fig. 10, namely a 4 byte Block Number field 28 (this is the block address of the COB or COSB) and a 32 byte Obsolete or Deleted Sector Mask 30 which is a BitMap containing ones in positions corresponding to obsolete or deleted sectors inside

5 this block. For a block comprising 256 pages this mask 30
takes 32 bytes.

Next Erased Block list (NEB)

10 5 The Next Erased Block list is created in the microprocessor
SRAM 13 in order to provide fast identification of the next
available erased block when incrementing the WP, SWP and RP
15 between blocks. Available erased blocks are used in ascending
order of their physical addresses. The NEB contains M erased
10 block addresses, (e.g. M=8). The NEB list is a list of next
available erased blocks, starting with the erased block with
20 block address closest to and higher than the address of the
last erased block to be allocated for use. Thus, although the
number of entries in the NEB is limited (e.g. to 8), the NEB
15 itself may contain information about more than the next eight
erased blocks.

The Next Erased Block list is derived from the MAP entry
30 (stored in CB) appropriate to the region of Flash memory
address space being accessed by the write pointers. This will
remain in SRAM as the active NEB until all erased blocks
defined by it are used, at which point it must be recreated
35 from the appropriate MAP entries. The CB contains sufficient
MAP entries to define the erase state of every block in Flash
25 memory. Both the NEB and the corresponding MAP entry are
updated by addition and removal of blocks during operation of
40 the memory system. An entry is removed from the NEB when an
erased block is allocated for sector or control data storage.
An entry is added to the NEB (unless the NEB is already full)
45 30 when an erased block is created at a block address which lies
within the range spanned by the other blocks in the NEB.

50 A single NEB entry defines a group erased blocks at contiguous
addresses, and defines the block start address (the first

block address in the contiguous group of blocks) and the group length (number of contiguous blocks in the group).

TBL Pointers (TBLP)

The TBL Pointers identify positions of TBL entries in the CB. They are also stored in the microprocessor SRAM and are used to provide fast sector address translation. The TBLP is created during initial scanning of the Control Block, and then is updated every time a new TBL entry is created in the CB.

ASB List (ASBL)

As aforementioned, the ASBL is created in SRAM 13 and supports fast address translation. It identifies the physical addresses of all ASB blocks and the SAT blocks with which they are associated. It is created during initialisation and has to be updated each time a SAT page write operation is performed. The ASBL entries are listed in order of previous accesses, that is, when an entry is accessed it is promoted to the top of the list. When a SAT block which does not currently have an associated ASB is accessed, an ASB is allocated and an entry for it is added at the top of the ASBL. The bottom entry of the ASBL, representing the least recently accessed ASB, is eliminated.

SAT Cache

A cache is maintained in the SRAM 13 for 32 contiguous SAT entries incorporating the entry most recently accessed from a SAT in Flash memory.

Capacity Map

Total Flash memory capacity in the memory system 10 is allocated to data and control structures as follows :

1. Logical Sectors

Capacity is allocated to storage of one valid data sector for each logical address within the declared logical

5 capacity of the card. This declared capacity is the
available physical capacity minus items 2 to 8 below, and
is defined by the formatter during card manufacture.

10 2. Boot Block

5 At least one block is allocated to the boot block.
Preferably, a second block is allocated for storing
another copy of the boot block.

15 3. Control Block

A number of blocks are allocated to storage of Control
10 Block entries (Signature, BBL, TBL and MAP). The fully
compacted Control Block occupies less than one block in
most cases. Additional blocks are allocated to the
20 Control Block for entries written between
compaction/rewrite operations.

25 4. Sector Address Table

This is the capacity allocated to the storage of the
blocks of the SAT. It is proportional to the logical
capacity of the card.

30 5. Additional SAT Blocks

20 A fixed number of blocks are allocated for ASBs to be
associated with defined SAT blocks.

35 6. Obsolete Sectors

One block is allocated for the COB. Another block is
allocated for the COSB and one further block is allocated
25 for the POB (for embodiments which allow the existence of
a COSB and POB). The maximum number of permitted obsolete
40 data sectors is therefore set by the number of pages in a
block.

45 7. Erased Buffer

30 This is a buffer of erased blocks which must be allocated
for correct operation of the system. At least one erased
block must be allocated for data sector relocation and
one for control structure relocation which may take place
50 concurrently.

35 8. Spare Blocks

Spare blocks may be allocated for use to maintain declared logical capacity in the event of a failure during operating life. The number of spare blocks is determined by the formatter during card manufacture.

5

A Capacity Allocation Table illustrating, for example purposes, capacity allocated to the above items 1-8, for an 8MB card, 64MB Card and 512MB Card (FLASH Cards) is shown in Fig. 16 (this shows only one block allocated for obsolete sectors, namely for the COB, but it will be appreciated that more blocks will be allocated for obsolete sectors where there is provision for a COSB and/or POB in the memory system).

SAT WRITE OPERATIONS

The SAT is rewritten when the WSL, WBL, SSL, SBL, RSL or RBL, is full and, in the majority of cases, it is done with a granularity of one page, that is, only SAT pages containing modified sector addresses are rewritten. However, in a small minority of cases, full SAT block rewriting is required (designated "SAT block write"). When a SAT rewrite is required, an Additional SAT Block (ASB) is created, into which only the required pages are written (designated "SAT page write"). There can only be one ASB dedicated to a specific SAT block, and totally, there may exist a limited number N of ASBs. The value of N will be chosen as an appropriate compromise between write performance and capacity overhead (for example, N=8).

Each ASB exists until a "SAT block write" is performed on its associated SAT block. A SAT block write is performed when a SAT write page is required on a SAT block whose ASB is full or an ASB is to be deallocated. When a SAT page write is required and the corresponding SAT block does not have an associated ASB and all N ASBs are already allocated, one of the existing ASBs has to be deallocated before allocation of a

5 new ASB. The ASB to be deallocated is selected as the one
which has not been written for the longest time. Note that
deallocation of an ASB is rather time consuming procedure as
10 it requires writing of a SAT block and also erasure of both
5 the obsolete SAT block and ASB.

Interleaved Chip Access

15 The above described operations and data structures inherently
allow interleaved write operations to be performed on several
10 Flash chips 5 and this can significantly increase performance.
The controller chip 8 may thus control a plurality of FLASH
20 chips 5, for example an array of four FLASH chips,
incorporated in the memory system. The memory space of the
four FLASH chips is organised by the controller as a set of
25 virtual blocks, each of which consists of four blocks 4, one
block from each of the four chips 5 (which chips are
permanently linked together). The four blocks in each virtual
30 block are the blocks with the same block address within a
chip. This organisation allows linked blocks forming a virtual
20 block to be treated as one large block and, therefore, to use
all the same described algorithms and data structures as
previously described for a single FLASH memory system 10,
35 using virtual blocks instead of the individual erasable blocks
of sectors. Page write operations are performed concurrently
25 across all interleaved chips. The Write Pointer (WP) and the
RP and SWP each move sequentially through page addresses, and
40 the ordering of address bits supplied to a hardware chip
enable decoder provided in the controller 8 ensures that pages
of linked blocks are sequentially accessed in the order
45 30 illustrated in Fig. 11 i.e. each pointer WP, SWP, RP moves
from one chip to another when moving from one PA to another
PA. This is achieved by the use of Virtual Addresses. A
50 unique Virtual Address (VA) is allocated to each physical
sector (e.g. in the above-described NAND based memory system a
35 VA is allocated for each page) in all of the chips. The

5 Virtual Addresses are allocated such that incrementing the VA
by one moves the write pointer from one chip to the next chip,
the Virtual addresses incrementing from chip to chip, though
10 linked blocks of each virtual block, in a repeating pattern as
5 shown in Fig. 11.

15 The controller in effect treats the array of memory chips as a
single column of virtual blocks. The Virtual Address of a
sector takes the format shown in Fig. 12. This consists of a
10 Virtual Block portion comprising a ChipHigh portion and the 13
bit Block address of the sector, and a Virtual Page portion
20 comprising the 6 bit page address and a ChipLow portion. The
ChipHigh portion is C_{high} bits of the 5 bit chip number (of the
physical address - see Fig. 5) and the ChipLow portion is C_{low}
15 bits of the 5 bit chip number, where:

C_{high} = column number of chip in array of chips; and

C_{low} = row number of chip in array of chips.

25 To obtain the Physical Address (PA) from the Virtual Address
(VA), the controller simply re-organises the VA so as to move
30 the ChipLow portion back between the ChipHigh and Block
address portion, as shown in Fig. 13. Thus, it will be
appreciated that in a single chip memory system for any sector
35 the VA is equal to the PA.

40 For simplicity, only a number of chips which is a binary
multiple may be interleaved, for example, 2 or 4. Erase
operations on virtual blocks are performed as concurrent
erasures of all linked blocks of interleaved chips. If a block
in a chip is a bad block the controller treats all the blocks
45 having the equivalent block address in the other chips as bad
blocks.

50 Interleaving is enabled or disabled according to the status of
a control byte in the Boot Block which is set in enabled or

disabled status by the manufacturer of the memory system when the FLASH memory is formatted.

Where block addresses or PAs were previously used in the above-described single chip NAND type FLASH embodiments, we now use Virtual Block addresses and VAs, respectively. On receipt of a host data sector write command the controller translates an incoming LA to a PA by allocating the PA to which the relevant write pointer is pointing. The controller controls the write pointers to each move through the PAs so as to, in effect, move sequentially through the Virtual Addresses (VAs) of the sectors of those of the virtual blocks which are erased (which erased virtual blocks are identified in the NEB).

Fig. 14 illustrates the timing of the various operations involved in a multiple sector write to interleaved chips. Fig. 14 will now be described with reference to Fig. 15 and Fig. 19. Fig 15 is a block diagram illustrating in detail the controller chip 8 of the memory system. (The controller chip 8 of Fig. 2 may be of the form shown in Fig. 15 and like parts in both Fig. 2 and Fig. 15 are numbered with the same reference numerals). Fig. 19 is a schematic diagram of a memory system 10 comprising the controller chip 81 and four FLASH chips 1¹, 2¹, 3¹, 4¹ each with its own read/write buffer 71¹, 72¹, 73¹, 74¹.

Fig. 15 shows the controller chip 8¹, with : an input/output port O/P e.g. a PC Card ATA Port, Compact Flash or IDE Port, for connection to a host computer; a Host Interface & Registers 80 connected to O/P and a Dual-Port SRAM Sector Buffer 9 connected thereto; A Datapath controller 82, the ECC Generator & Checker 12, and a Flash Memory Interface (FMI) 84, all connected to the Sector Buffer 9, the FMI also being connected to a Flash Memory Port 86. The controller 8 also

5 includes microprocessor 11 (in the form of a RISC Processor);
processor SRAM 13,; a processor Mask ROM 88; and a port for an
external Program RAM or ROM 92. An optional Debug Port 94
10 may also be provided for the RISC processor 11. Data and
5 commands are communicated between the various components of
the controller 8¹ (except for the Sector Buffer 9) via a
microprocessor Bus 91.

15 As shown in Fig. 14, when a multiple sector write command (in
10 this case a 4-sector write comprising Host Data sectors 1, 2,
3 & 4) is received at the ATA Port O/P, Sector 1 is written
20 into a Buffer 1 of the Dual-Port Sector Buffer 9. This leaves
a Buffer 2 of the Sector Buffer 9 available for controller
data management operations. When Sector 2 is received it is
25 written directly into Buffer 2 and at the same time Sector 1
is moved from Buffer 1 to the Flash Memory Port 86 from where
it is written into the read/write buffer 71¹ of one of the
four FLASH chips (chip 1¹). Sector 2 is then sent from buffer
30 2 to the Flash port 86 and on to the read/write buffer 72¹ of
20 one of the other four FLASH chips (chip 2¹). While this is
happening Sector 3 is received directly into Buffer 1 of the
Sector Buffer 9. Sector 3 is written to the buffer 73¹ of
35 Chip 3 and Sector 4 is received into Buffer 2 of the Sector
Buffer 9, and is then written to the buffer 74¹ of chip 4.
25 Sectors 1,2,3 and 4 are then written into the relevant
allocated physical sectors in the memory arrays 61¹,62¹,63¹,64¹
40 of chips 1¹, 2¹, 3¹ & 4¹ respectively. Although Fig. 14 shows
each such sector Write operation starting shortly after the
previous one, in practice, to all intents and purposes, it
45 30 will be appreciated that the four Sectors 1,2,3,4 are written
substantially concurrently to the Flash Chips 1¹ to 4¹.
Moreover it will be appreciated that the physical addresses of
the sectors to which the Host Data Sectors 1 to 4 are written
50 are determined by the algorithms previously described which

determine the position of the relevant write pointer (i.e. sequential use of Virtual Addresses).

It will be appreciated that where a multiple-sector write command of more than four sector writes is sent from the host processor to the controller, the controller partitions the multiple-sector write into groups of (for the present embodiment using four memory chips) four sectors, each such group to be written to FLASH memory in an interleaved write sequence as described above with reference to Fig.14.

Address Translation

The process of address translation for an LA to a VA will now be described in further detail, with reference to read and write operations.

Address translation is an operation performed on the logical address of a sector which returns one of the following three values:

- Valid sector physical address
- Information that logical sector has been deleted (or has never been written)
- Information that an error condition has occurred

Fig. 17 is a flow chart illustrating the address translation process. This process is carried out for every read operation. When a logical sector address (LA) to be translated is received by the controller 8¹ from the host processor an algorithm is implemented (box 40) to identify the possibility that a sector whose logical address is to be translated has previously been written or deleted since its SAT entry was last written. A conclusion that a sector has not been previously written or deleted must be 100% certain; a conclusion that a sector may have been previously written or deleted should have a high probability of being correct. This

5 is carried out by the processor by keeping a limited number of
pairs of values representing the start and end addresses of
contiguous sequences of sectors, and identifying whether an
10 address to be translated lies within any of these ranges.
5 These ranges may eventually become non-contiguous, leading to
some uncertainty in a conclusion that a sector may have been
previously written or deleted. If the LA lies within any of
15 the ranges then we answer "IS Repeat Possible?" (box 42 of
Fig.17) with YES. If the LA does not lie within the ranges we
10 answer NO and go to the SAT or SAT Cache to find VA (box 44).
From here we determine whether the physical sector is Bad (46)
20 or Deleted (50). If the LA corresponds to an unwritten sector
this results in VA=Deleted (58) at box 50. If we answer Yes
at box 42, then we search 52 the WSL or SSL (depending on
25 whether the LA corresponds to file or system data). If at box
54 the LA is found (YES) the VA is calculated 56 and the
logical address stored in the header 1b is of the physical
sector is read by the controller microprocessor (at 58). If
30 LA=LA1 (box 60) then the calculated VA is correct. If the LA
20 is not found at 54 then we search 62 for it in the RSL and if
it is not found in the RSL we go to the SAT or SAT Cache and
get the VA from there 44. If the VA found in the SAT or Sat
35 Cache is not Bad or Deleted 46, 50, then we get LA1 from the
VA 58 and check if LA=LA1, as before.
25
40 The process steps carried out at box 56 (Calculate VA) of
Fig.17 are illustrated in detail in the flow diagram of Fig.
20. Fig.20 illustrates the steps carried out in order to
obtain the VA for an LA found in the WSL or RSL, for a memory
45 30 system using only two write pointers, WP and RP. It will be
appreciated that this flow diagram would be extended to also
allow the VA for an LA found in the SSL, where the memory
system also incorporates a System Write Pointer (SWP). The
50 process starts at box 100 where we set NumFromEnd (NFE), where
35 NumFromEnd = number of sectors written beginning from the end

5 of the WSL (or RSL) up to the given sector (found in the WSL
or RSL. If the LA was found in the WSL we set $P=WP$ and if LA
was found in the RSL we set $P=RP$; then we set $PG= P.Page$ where
10 $P.Page$ is the page of the write pointer indicated by the value
5 of P (see box 102). If $104 PG > NFE$ (i.e. LA is in the last
written block) then $VA= P-NFE-1$, namely $NFE-1$ sectors away
from the position of the relevant write pointer. If $PG < NFE$
15 then we determine 106 if $P==0$, namely if a block corresponding
the last WBL/RBL entry is fully written. If it is (i.e. $P==0$)
10 we set $NotLast=0$, and if it is not we set $NotLast=1$. We then
calculate 108 number of blocks, $Nblock$, between the last one
20 and the block where the given sector lies, using the following
algorithm:

$Nsect$ is a number of sectors between last written block page 0
25 and the given sector;
 $Nsect= NumFromEnd - PG$;
 $Nblock= Nsect/BlockSize + NotLast$

30 We then calculate 110 page number in a block, $PageNum$, where
20 $PageNum= BlockSize - Nsect\&BlockSize$. If LA is in the WSL we
then 112 get Block Address ($BLAddr$) from the WBL, or if LA is
in the RSL we get 114 Block Address ($BLAddr$) from the RBL,
35 where Block Address is a Virtual address of a block containing
the given sector, using the following:

25 If LA is I WSL, then $BLAddr= RBL[LBL - Nblock]$, where LBL is
an index of the last entry in the WBL;
40 If LA is in RSL, then $BLAddr= RBL[LRBL + Nblock]$, where $LRBL$
is an index of the last entry in the RBL.

45 30 Then we calculate 116 the VA using: $VA= Page0 + PageNum$, where
 $Page0=$ Virtual address of page 0 in the block containing the
given sector.

50 Fig.21 is a flow diagram of the process used to Get VA from
35 SAT or SAT Cache (box 44 in Fig.17). Fig.21 is generally self-

5 explanatory but further comments regarding specifically
labeled boxes is given as follows:

10 Box 120 (Is LA in SAT Cache): LA is in the SAT Cache if $LA \geq$
5 FirstCacheEntry.LA $\leq LA < \text{FirstCacheEntry.LA} + \text{CacheSize}$,
where FirstCacheEntry.LA = the LA corresponding to the first
SAT Entry in the Cache, and (global) CacheSize = number of
15 entries in the SAT Cache;

10 Box 122 (Calculate Block and Page in SAT): We calculate SBNu
which is a SAT block number for the given LA, and Spage which
20 is a SAT page number for the given LA;

Box 124 (Calculate number of TBL entry): TBLNum is a number of
25 15 the required TBL Entry, where $\text{TBLNum} = \text{SBNu} / 8$; and

Box 126 (Store part of SAT page in cache): If it is possible,
32 entries starting with the last entry accessed are cached.
30 If there are not enough entries then a group of 32 entries
20 ending with the last entry in a page and including the last
sector accesses is cached.

35 Fig.18 is a flow diagram illustrating the process steps for
Box 58 (Get LA1 from VA) of Fig.17. It should be noted that
25 the Header Parameter (HP) stored in the Page Header will be
40 the value of the logical address (LA) incremented by one. This
is because deleted sectors are marked by setting all bits in
their headers to zero. This LA=0 cannot be stored in a header.
We therefore set $\text{LA1} = \text{HP} - 1$.

45 30

Read Operations

Fig.22 is a flow diagram illustrating the sequence of steps
carried out to read a host data sector from a physical sector.
50 The controller starts by translating the LA (received from
35 the host) to a VA (box 130). This is done by carrying out the

5 process illustrated in the Fig.17 flow chart, already
described. Once the LA has been translated to a VA, the
content of the physical sector with address VA is read into
the buffer 9 of the controller. The controller then checks
10 (box 132) if the sector (i.e. the content of the sector) is
deleted or bad. If it is deleted or bad, the controller sets
all bytes of the host processor's data buffer to 0xFF (box
15 134). If the sector is not a deleted sector (box 136), the
controller returns error status to the host (box 138). If the
20 sector is a deleted sector, the controller returns valid
status to the host (box 137). If at box 132 the controller
determines that the sector is not deleted or bad, the
controller goes straight to box 137 i.e. returns valid status
to the host.

15

Write Operations

Fig.23 is a flow diagram illustrating the sequence of steps
carried out to write a host data sector to a physical sector.
30 Fig.23 deals only with write operations for host file data,
and thus written by the Write Pointer (WP), but it will be
appreciated that the operations of Fig.23 would be
appropriately extended to deal with separate system data
35 writes where the memory system uses a separate write pointer
for system data (i.e. the SWP). The controller starts by
25 translating the LA (received from the host) to a VA (box
150). This is done by carrying out the process illustrated in
the Fig.17 flow chart, already described. If 152 the sector is
Bad, the controller returns error status 154 to the host. If
the sector is not Bad, then check if the sector is deleted 156
45 30 and if it is deleted then check if WP is valid or invalid 158.
WP is invalid (WP==0) when a full block has just been written
and WP has to be moved to an erased block. If WP is not valid,
we set the WP to a new (valid) physical sector address 160.
50 When WP is valid, we add the LA to the WSL 162 and perform any
35 WSL or RSL release, and/or CB and CBPT compaction, which is

5 necessary. We then update 164 the ranges from the Evaluate
Repeat Possibility algorithm (box 40 of Fig.17) and write the
sector 166 from the controller buffer to the address of the
WP, and return a valid status value 168 to the host. If, at
10 5 box 156, we find the sector is not deleted we then check 157
if the VA is in the COB. (The VA is in the COB if VA coincides
with VB, where VB is a Virtual Block Number field (this is the
Virtual Block address - see Fig.12) in the COB structure
15 stored in SRAM 13. If the VA is in the COB, we record VA as
10 obsolete 159 in the COB structure stored in the controller
SRAM 13 (this is done by setting a corresponding bit to 1 in
the BitMask field of the COB Structure in SRAM 13.) and then
20 go on to box 158 (is WP valid). If the VA is not in the COB we
change the COB 161 and then move on to box 159 (Mark VA as
15 Obsolete).

25 Fig.24 is a flow diagram of the steps implemented at box 161
of Fig.23 (Change the COB). Fig.24 is generally self-
explanatory but should be read in conjunction with the
30 20 following notes:

35 Box 200 (VA.B1, VB): VA.B1 is a Virtual Block field of VA, and
VB is as above-described;

Box 202 (Is COB invalid): COB is invalid if VB=0. If VB is
25 equal to zero this indicates that there are no obsolete data
at this moment;

40 Boxes 203,204 (Calculate MaxRel): MaxRel is a maximum number
of sectors to be relocated from COB. MaxRel=P.Page-1, where
P.Page is a Page field (address) of the WP or RP;

45 30 Boxes 205,206 (Add Dummy Entries to WSL): If a block to be
relocated is not fully written yet, corresponding "dummy"
sector LAs must be added to the last WSL (RSL) Entry;

50 Box 207 (Relocate Sectors): See Fig.25;

5 Box 208 (Write WMAP): Write WMAP to the CB where EB+VB and
corresponding bit in the BitMap is set to 1. Perform CB
rewrite if necessary;

10 Box 209 (Update Lists): Find WRBArray entry equal to VB and
5 mark it and any other entries for the same VB in WRBArray as
invalid. The WRBArray is in fact the WBL and WSL lists which
are actually stored in the same area of memory with the WBL
15 entries at the start counting up and the RBL entries at the
end counting down. The WRBArray is full when the two lists
20 meet in the middle.

 Box 210 (Setup COB): Update COB Structure in SRAM, VB field is
20 set to VA.B1, Obs and Del Mask bit corresponding to VA.Page is
set to 1, all other bits are set to 0.

25 Fig.25 illustrates the steps implemented at box 207 (Relocate
Sectors) of Fig.24. This should be read in conjunction with
the following notes:

 Boxes 220, 222 and 230: Perform a loop going through Obs and
30 Del Mask field of the COB Structure in SRAM;

20 Box 223 (Is Sector Valid?): Sector is valid if ODMask[i]=0;
zero value in COB Obs and Del Mask indicates that this page
contains a valid sector;

35 Boxes 224 and 225: RP=0 if a block pointed to by the RP is
already fully written;

25 Box 226 (Store LA from Page Header): LA got from Page Header
is temporarily stored to be used in Add Entry to RSL;

40 Box 225 (RP=0): Add Entry to the RSL. Perform WSL/RSL release,
if necessary.

45 Fig.26 illustrates the steps implemented in order to set the
Write Pointer (WP), at box 160 in Fig.23. A similar process is
used to set the RP at box 227 in Fig.25. Fig.26 should be read
in conjunction with the following comments:

Box 240 (Is WRArray not full): WRArray is not full if
Last<LastRE-1, where Last (global) is an index of the last WSL
entry, and LastRE is an index of the last RSL Entry;

Box 242 (Is WRBArray not full): WRBArray is not full if
5 LBL<LRBL-1, where LBL and LRBL are as previously defined with
regard to Fig.20;

Box 244 (Lists Release): Perform SAT Page Write operation, SAT
Block Write and CB rewrite if necessary;

Box 246 (Fill NEB): Select next N (N=NEBSize) erased blocks
10 from MAPs stored in the CB;

Box 248 (Write WMAP): Write WMAP with Link field set to
ErBlock and corresponding bit in BitMap field set to 0.
Perform CB rewrite if necessary.

15 Fig.27 is a flow diagram of the process steps carried out in a
delete sector operation. This is in the initial steps similar
to a write operation (see Fig.23): the LA is translated 250 to
a VA. If 252 the sector is deleted we return error status to
the host 254. If the sector is not deleted we check 256 if the
30 VA is in the COB and if it is not 261 in the COB we Change the
COB 261 (same as Fig.24). If VA is in the COB we Mark VA in
the COB as deleted 269, then we fill one of the buffers of the
Dual port SRAM 9 (in the controller) with zeros 271, and then
35 write this "all zeros" page from the buffer to the sector to
be deleted (thereby deleting the sector). We then return valid
status 275 (confirming sector has been deleted) to the host.

Initialisation

Initialisation procedures and power-loss recovery procedures
45 will now be described. For simplicity, these are described
with reference to a single write pointer system (i.e. only WP)
but it will be appreciated that the procedures will be readily
extended as appropriate for the multiple write pointer system
50 (WP, SWP, RP).

5 All data and control structures are specially constructed to
avoid generalised scanning during initialisation. Almost all
control structures (except WSL and WBL) normally are derived
10 from corresponding information stored in the CB. During
5 initialisation of the card, it is necessary to perform
following operations.

- 15 1. To read the last Control Block Pointers Table entry from the
Boot Block and so identify the CB block(s) locations.
2. To reconstruct the TBLP by scanning the CB.
- 10 3. To scan header/ECC fields of pages sequentially following
the write pointer position defined in the last TBL entry in
20 the Control Block to identify sectors written since the last
SAT rewrite and to construct the WSL and WBL.
4. To construct the NEB from corresponding MAP entries in the
25 CB.
5. To construct the COB and ASBL.
6. To check if a block referenced in the ErB field of the last
MAP is really erased. If not, to complete erase operation.

30 20 Construction of WSL and WBL

During initialisation of the card, the last value of the Write
Pointer (WP) to be stored is read from the latest TBL entry in
35 the CB and a scan of page headers at successive pages from
that location is performed to rebuild the WSL and WBL in
25 processor SRAM. When an erased location is encountered, the
end of the sequence of sectors written since the last SAT
40 rewrite has been reached.

This sector scan must take account of the fact that the Write
45 30 Pointer (WP) may jump from the end of a block to the beginning
of a non-adjacent block. All block transitions made by the
(WP) are recorded in the Link fields of MAP entries in the CB.

50 Construction of COB and ASBL

5 These structures can be reconstructed by copying corresponding
entries from the CB. In addition, to construct the COB
structure (to identify deleted sectors in it) it is necessary
10 to scan a current block containing these sectors, whose
5 address is defined in the Obs field of the last MAP entry in
the CB. To identify obsolete sectors in this block, it is
also necessary to scan WSL and WBL. In order to record ASBL
15 pages we have to identify ASB addresses from the TBL and then
to scan their header/ECC fields.

10

Power-Loss Recovery

20 It is a requirement for the memory system that it should be
able to operate normally, and that no stored data should be
lost, when power is restored, whatever the circumstances under
25 which power has been removed. However, it is not necessary to
restore the full normal state of the memory system immediately
after power-on, only to allow it to operate normally... A
normal state can be restored later as an exception, whenever
30 any abnormal state is detected.

20

The normal state of the Memory System may be degraded if the
supply voltage is removed whilst any of the following
35 operations is being performed.

1. Writing of a data sector from a host
- 25 2. Writing of a data sector which is being relocated
3. Writing of an entry to a control data block (CB or BB)
- 40 4. Writing of a page to a control data block (SAT or CB)
5. Erasure of any block with obsolete sector or control data

30 Power loss during writing of a data sector from a host

In this case, the data being written may be lost, but the host
had not been informed that the write command had completed and
may write the sector again. An incompletely written sector
50 may exist in Flash memory as a result of the partial write
35 operation. This is detected during initialisation when the

5 value of the Write Pointer is established by reading the page
headers in the block defined by the last Link parameter in the
CB. The last detected sector should read fully to check its
10 ECC, and the next page should be read to check that it is
5 completely erased. If a partially written sector is detected,
all other sectors should be relocated to a new Write Pointer
position at the beginning of the next erased block, then the
15 block should be erased.

10 Power loss during writing of a data sector which is being
relocated

20 This is detected during the process of establishing the Write
Pointer during initialisation, as above. The same action of
relocating sectors and erasing the block should be taken. In
25 addition, an incomplete relocation operation should be
detected by comparing logical sector addresses immediately
preceding the Write Pointer with those of obsolete sectors in
the block defined by the Obs parameter in the CB. Any pending
30 sector relocations should be completed during initialisation.

20

Power loss during writing of an entry to a control data block
(CB or BB)

35 This condition may be detected during normal initialisation
when entries in the CB and BB are read and their ECCs are
25 checked. The entry should be ignored if its ECC shows an
error. The earlier operation which initiated the CB or BB
40 entry write operation had not completed correctly and the
entry will later be written correctly when this operation is
repeated during normal operation.

45 30

Power loss during writing of a page to a control data block
(ASB)

50 This condition may be detected during normal initialisation
when pages in the ASB are read and their ECCs are checked.
35 The page should be ignored if its ECC shows an error. The

55

earlier operation which initiated the ASB page write operation has not completed correctly and the page will later be written correctly when this operation is repeated during normal operation.

Power loss during writing of a full control data block (SAT or CB)

This will result in an incomplete control data block existing in Flash memory, with no references to it by other data structures. This condition need not be detected during initialisation, and the block may be allowed to exist as a "lost block". The earlier operation which initiated the block write operation had not completed correctly and the block will later be written correctly when this operation is repeated during normal operation. At a later stage of normal operation, the lost block will be detected by a discrepancy with its MAP state, or by the discovery of a discrepancy in the number of erased blocks in the system (see Capacity Map in Fig. 16). Exception routines may then identify and erase the block, by full FLASH memory scanning if necessary.

Power loss during erasure of a block with obsolete sector or control data

This will result in an incompletely erased block existing in Flash memory. This condition is detected during initialisation when the state of the block referenced by the ErB field in the last MAP entry in the CB is checked. Re-erasure of this block can be performed, if necessary.

Further Alternative Embodiments

Various modifications to the above-described embodiments are possible without departing from the scope of the invention. For example, one alternative way of handling erasure operations is to always allow two COBs (and two COSBs) to exist: the advantage of this would be to make the best use of

5 memory capacity. In the above-described embodiment, we only
allow one COB, but also allow a POB to exist temporarily when
there is a write pointer in a block which we wish to make the
COB. This means that there must at all times be enough erased
10 5 memory capacity to allocate for a POB, should it be necessary
to have a POB. It therefore is attractive to make the best use
of this memory capacity and one way of ensuring this is to
always allow two COBs to exist, therefore eliminating the need
15 for a POB (the second COB can act as a POB, when required). In
such a two COB system, when it becomes necessary to create a
new COB we erase the older one of the two COBs (unless it has
20 a write pointer in it in which case we erase the younger one).

With reference to Fig.3, and the description of the
25 15 arrangement of data in each of the FLASH pages in the memory
system, we also propose some alternative ways of storing the
data within a page. Fig.28 shows the physical partitioning of
a typical 528 Byte NAND or AND type FLASH memory page 1. The
30 page comprises a 512 Byte "data area" 300 and a 16-Byte "Spare
20 area" 302. In the embodiment described above with reference to
Fig.3, the controller 8 stores 512Bytes of information 1a
(e.g. one host data sector) in the Data Area 300, and stores
35 the Header 1b and ECC 1c (together referred to hereinafter as
Overhead Data (OD)) in the Spare Area 302. However, other
25 arrangements of data within the page 1 are possible. For
example, as shown in Fig.29, the Header 1b and ECC 1c could
40 equally be stored in a first portion 303 of the Data Area 300,
and the Information 1a stored in the portion 304 consisting of
the Spare Area 302 and the remainder of the Data Area 300.

45 30 Another possibility, shown in Fig.30, is to write the Header
1b and ECC 1c at a position offset from the start of the FLASH
page, and to write the Host Data Sector (which may be referred
50 to as the "user data") in the remaining space either side of
35 the Header and ECC. By how much (Offset S) the OD is offset

5 may, for example, be determined by a function of either: (a) the physical address (PA) of the page 1; or (b) one or more bits within the first byte of user data (i.e. the host data sector) written to the page 1. Fig.31(a) illustrates the
10 arrangement of data in the controller buffer 320 before the start of a sector write operation, the data being arranged as a first portion 322 of user data and a second portion 324 of Header data. Fig.31(b) shows the arrangement of the data in a FLASH memory page, following completion of the write operation
15 in which the offset S is determined by one or more bits within the first byte of user data (option (b) above). The data is stored as a first portion 326 of the user data, followed by a second portion 328 of the user data, followed by the Header 1b and ECC 1c, followed by the third and final portion 330 of the
20 user data. The length of portion 326 + portion 328 is dependent on data within portion 326. The length of portion 326 is defined to be less than or equal to the minimum offset, and the length of 328 is calculated on the basis of data within portion 326 to provide the correct Offset S. The first
25 and second portions 326, 328 of user data are separately identified so that the first portion 326 may be read from the FLASH memory by the controller in one operation, and evaluated by the controller in order to determine the length of the second portion 328 which should be read by the controller in a
30 subsequent operation. Fig.32 is a table detailing the sequence of controller commands used to transfer the data from the controller buffer to the FLASH memory during the write operation.

35 One advantage of choosing the offset S to be a function of one or more bits of the user data is that the overhead data is therefore not inserted at the same position in the 528Byte data segment in every sector. This protects the valuable
40 overhead data from simultaneous loss on a large number of
45
50
55

sectors in the event of a systematic failure in FLASH memory, such as a memory array column failure.

Fig.33 shows the resulting arrangement of data in the controller buffer after completion of a read operation on the FLASH memory page of Fig.31(b). From Fig.33 it will be seen that the data in the buffer is arranged back to a first portion 322 of user data and a second portion 324 of Header data, now followed by a third and final portion 325 of ECC. Fig.34 is a table detailing the sequence of controller commands used to transfer the data from the FLASH memory to the controller buffer during the read operation.

Additionally, with reference to interleaved write operations to multiple FLASH chips, as described already with reference to multiple FLASH chip memory systems, we also propose that this technique for writing substantially concurrently to a plurality of chips may also be used for writing data to a single memory chip in which the physical page size is a multiple of the size of a sector write by the controller e.g. each page of the memory is four times the size of a segment of (user + overhead) data written by the controller, where the controller writes data in uniformly sized segments.

It will further be appreciated that the invention is applicable not only to NAND-type memories but also to other types of memory, such as AND and NOR type memories. In the case of AND type FLASH memory, each page 1 of a block has the same format as the NAND page format of Fig.28 and we can use any of the possible arrangements of data within the pages as afore-described. We design the controller to still erase memory in blocks of sectors, although in blocks containing bad sectors the individual good sectors in the block to be erased will be erased individually. Thus, the controller does not treat any blocks containing bad sectors as bad blocks, but

5 instead treats them as good (erasable) blocks and makes use of
the good sectors within those blocks. In AND type embodiments
though we ensure that the controller only uses blocks
10 containing all good sectors for SAT blocks or ASBs.

5

Where AND type FLASH memory is being used and the memory
system is a multiple FLASH chip system utilizing interleaved
15 chip write operations as described above, where any block of
sectors (pages) in one of the virtual blocks contains a bad
10 sector, the controller causes the write pointers to skip this
sector and go to the next good sector in the block e.g. where
20 c=chip and s=sector, if a burst of four sector writes is c3s5,
c4s5, c1s6, c2s6 then if c1s6 is a bad sector the sequence
becomes c3s5, c4s5, c2s6, c3s6. This is in contrast to
25 15 embodiments based on NAND type memory, where if one block in a
virtual block contains one or more bad sectors the controller
treats that block as a bad block and treats the whole virtual
block as a bad virtual block.

30 20 Where we use NOR type FLASH memory, our preferred embodiment
is one in which we design the controller of the memory system
to still read and write data structures to and from the FLASH
35 memory in uniformly sized sectors, each sector being 528 Bytes
wide. Fig.35 illustrates schematically three such sectors
25 1,2,3 in a block 4' of NOR memory. Due to the fact that one
row of memory in a NOR block is only 512 Bytes wide it will be
40 appreciated that each of our sectors in NOR therefore fills
one row and wraps round to fill a part of the next row.
Nevertheless, it would be possible to define our sectors in
45 30 NOR memory in a different manner, and we may choose to use
sectors of smaller or larger size than 528 Bytes, and a block
could even contain sectors of more than one size. The
50 controller may arrange the data within each sector in any of
the various different ways already described with reference to

5 NAND and AND type memory pages, each sector including user and overhead data.

10 It will be appreciated from the foregoing that the physical
5 sectors of the memory system, whether the memory system is
based on NAND, AND or NOR type memory arrays, need not have
any particular relationship to the physical architecture of
15 the memory array itself, for example a sector need not
correspond to a row (page) of the array, and no physical
10 feature need be present to identify the boundary between one
sector and a neighbouring sector. A sector can be interpreted
20 as a group of memory cells which are always treated by the
controller as a unit. Different sectors need not be of the
same size. Moreover, the physical structure of a sector has no
25 dependence on data which might be stored in the sector. Also,
embodiments are possible in which defective regions within a
row (page) of memory cells are tolerated and are simply
skipped over by the controller when writing to physical
30 sectors.

20
With reference to the SAT, while as above-described the SAT is
preferably stored in one or more blocks of the memory array 6,
35 it would alternatively be possible to provide in the memory
system 10 a separate non-volatile memory which is accessible
25 to the controller, in which separate memory the controller
40 stores the SAT.

Finally, in a modified version of the above-described
embodiment, instead of always using available erased blocks in
45 30 ascending order of their physical addresses as above-
described, the controller uses the erased blocks in another
order. In this modified embodiment, the NEB list contains a
50 chosen subset of all the currently available erased blocks,
the first block address in the NEB list is the next erased
35 block to be used, and this first block address is removed from

5 the NEB list when it has been allocated for data storage use.
Any new erased block which is created (e.g. due to creation of
obsolete data, following a delete command from the host) is
10 added to the bottom of the NEB list. This continues for a
5 period determined by the controller (which could be a
predetermined number of sector write commands from the host,
for example), at the end of which period the controller re-
15 compiles the NEB list by replacing the entries in the NEB with
a new subset of the currently available erased blocks.
10 Conveniently, subsets of the whole set of all erased blocks
may be used sequentially in order of ascending physical block
20 addresses. This modified embodiment may have some advantage in
reducing memory space requirements in connection with
monitoring and storing the erased state of all blocks.

Claims

5

10

15

20

25

30

35

40

45

50

55

5

CLAIMS

10

1. A memory system for connection to a host processor, the
5 system comprising:

15

a solid state memory having non-volatile memory sectors which
are individually addressable and which are arranged in
erasable blocks of sectors, each said sector having a physical
address defining its physical position in the memory;

20

10 and a controller for writing data structures to and reading
data structures from the memory, and for sorting the blocks of
sectors into blocks which are treated as erased and blocks
which are treated as not erased; wherein the controller
includes:

25

15 means for translating logical addresses received from the host
processor to physical addresses of said memory sectors in the
memory;

30

a Write Pointer (WP) for pointing to the physical address of a
sector to which data is to be written to from the host

35

20 processor, said Write Pointer (WP) being controlled by the
controller to move in a predetermined order through the
physical addresses of the memory sectors of any block which is
treated as erased and, when the block has been filled, to move
to another of the erased blocks;

40

25 wherein the controller is configured so that, when a sector
write command is received from the host processor, the
controller translates a logical address received from the host
processor to a physical address to which data is written by
allocating for said logical address that physical address to

45

30 which said Write Pointer (WP) is currently pointing, and
wherein the controller is configured to compile a Sector
Allocation Table (SAT) of logical addresses with respective
physical addresses which have been allocated therefor by the
controller, and to update the SAT less frequently than memory

50

35 sectors are written to with data from the host processor.

55

5 2. A memory system according to claim 1, wherein said Write
Pointer (WP) is controlled by the controller to move in a
predetermined order through the blocks which are treated as
10 erased.

5 3. A memory system according to claim 1 or claim 2, wherein
the physical sector addresses in the SAT are ordered by
15 logical sector address (LSA), whereby the Nth SAT entry
contains the physical address of a sector to which data having
10 logical address N has been written.

20 4. A memory system according to claim 3, wherein the
controller is configured so that when a sector read command is
received from the host processor the controller looks up a
25 15 logical sector address (LSA) received from the host processor
in the SAT in order to obtain the physical sector address
which the controller previously allocated to said logical
sector address.

30 5. A memory system according to any preceding claim, wherein
the SAT is stored in at least one of said blocks of memory
sectors in the solid state memory.
35

6. A memory system according to claim 5, wherein the
25 controller is configured to update the SAT by rewriting the
SAT in whole blocks.
40

7. A memory system according to claim 5 or claim 6, wherein
there is provided at least one block (ASB) of sectors
45 30 containing modified versions of individual sectors of a SAT
block.

8. A memory system according to claim 7, wherein each sector
50 in a said ASB block contains the physical address of the

sector of the SAT block which it updates, and the modified version of the said sector of the SAT block.

9. A memory system according to claim 7 or claim 8, wherein when all the sectors in a said ASB block are written to with modified versions of SAT sector(s), the respective SAT block is rewritten so as to include all the modified versions in the ASB block and the ASB block is erased.

10. A memory system according to any preceding claim, wherein the controller is configured to control the Write Pointer (WP) so as to move sequentially, in ascending numerical order of physical address, through the erased blocks, as each block is filled with data written thereto.

11. A memory system according to claim 10, wherein the control of the Write Pointer (WP) is cyclic in the sense that once the sectors in the highest block, according to physical address order, have been filled with data the WP is controlled by the controller to wrap around to the block of sectors having the numerically lowest physical block address out of all the blocks currently being treated by the controller as erased.

12. A memory system according to any of claims 1 to 9, wherein the controller is configured to control the Write Pointer (WP) to move non-sequentially, according to physical address order, through the erased blocks.

13. A memory system according to any of claims 1 to 12, wherein each said memory sector (1) is physically partitioned into a data area (300) and a spare area (302) and the controller is configured so as to write overhead data (OD) comprising header data and error correction code data (ECC) at a position in the sector which is offset from the start of the data area (300) of the sector and to write user data, received

5 from the host processor, in the space remaining in the sector,
on either side of the overhead data (OD).

10 14. A memory system according to claim 13, wherein said
5 overhead data (OD) is offset by an amount which is determined
by at least one bit of the user data to be written to the
sector.

15 15. A memory system according to any preceding claim, wherein
10 the memory sectors in each said block of sectors are erasable
together as a unit.

20 16. A memory system according to claim 16, wherein the memory
sectors in each said block of sectors are also individually
15 erasable.

30 17. A memory system according to any preceding claim, wherein
the controller is configured to control erase operations on
the memory so as to only erase whole blocks of memory sectors,
20 and wherein a block of sectors is treated by the controller as
an erased block if all the memory sectors therein are erased
sectors.

35 18. A memory system according to claim 17, wherein if a block
25 contains one or more bad sectors, the controller defines the
whole block as being bad and treats that block as a not erased
block, whereby no data will be written thereto.

45 19. A memory system according to claim 16, wherein if a block
30 contains one or more bad sectors the controller treats that
block as an erased block whereby the controller may still use
good sectors in the block to store data, and wherein the
50 memory system includes a table identifying bad sectors and the
controller is configured to check whether the next sector
35 address to which the Write Pointer (WP) is to be moved is the

5 address of a bad sector and, if it is the address of a bad
sector, to control the Write Pointer to skip this bad sector
and move to the next sector address according to the
predetermined order in which the sectors are to be written to.

10 5

20. A memory system according to any preceding claim, wherein
each block of sectors has a physical block address defining
its physical position in the memory and the physical address
15 of each said memory sector includes the physical block address
10 of the block in which it is located, and wherein the
controller is configured to compile a list of the physical
block addresses of at least some of the blocks of sectors
being treated as erased, listed in an order in which the WP, is
20 to move through the blocks, which list is used by the
15 controller in order to quickly identify the next block of
sectors to be written to, and the memory system further
includes temporary memory means in which said list is stored
by the controller.

30 21. A memory system according to any preceding claim, wherein
the controller is configured so that, when a sector write
command is received by the controller from the host processor
35 which command renders obsolete data previously written to
another sector, the controller stores in a temporary memory of
25 the memory system the address of the sector containing the now
obsolete data.

40 22. A memory system according to claim 21, wherein the
controller is further configured so that if a sector delete
45 30 command, generated by a user, is received from the host
processor by the controller, the controller marks as obsolete
the sector to be deleted and stores the address of the sector
in said temporary memory.

50

55

5 23. A memory system according to claim 21 or claim 22, wherein
the controller is configured so as to allow only a fixed
predetermined number of blocks at any time, herein referred to
as the Current Obsolete Blocks (COBs), to contain one or more
10 5 sectors containing obsolete data which was written by the
Write Pointer (WP), and so that when all the sectors in a said
COB contain obsolete data, the said COB is immediately erased.

15 24. A memory system according to claim 23, wherein the
10 controller is configured so that where a sector in a block
other than a said COB is to contain obsolete data, the
controller: relocates any data in valid (not obsolete) sectors
20 in a said COB to another block and then erases the said COB;
marks said sector in the said block other than a COB as
15 obsolete; and designates said other block as a new COB.

25 25. A memory system according to claim 23 or claim 24, wherein
said fixed predetermined number of COBs is one.

30 26. A memory system according to claim 24, wherein said block
to which the controller relocates said valid data is the block
in which the WP is currently located.

35 27. A memory system according to claim 24, wherein the memory
25 system includes a further write pointer, herein referred to as
the Relocation Pointer (RP), for pointing to the physical
40 address of the sector to which said valid data is to be
relocated, the RP always being in a different block of sectors
to the Write Pointer (WP).

45 30 28. A memory system according to claim 27, wherein the memory
system includes a further write pointer, referred to as the
50 System Write Pointer (SWP), which points to the physical
address of the sector to which system data is to be written

5 from the host, the SWP always being in a different block to the Write Pointer (WP).

10 29. A memory system according to claim 28, wherein the controller is configured so as to allow at least two blocks which contain one or more obsolete sectors to exist at any time, one being said COB and the other being a Current Obsolete System Block (COSB) containing one or more obsolete system data sectors and, if any system data sectors need to be
15 relocated in order to allow the COSB to be erased, the relocated system data is sent to the address to which the System Write Pointer (SWP) is currently pointing.
20

25 30. A memory system according to claim 28, wherein the memory system includes another write pointer, herein referred to as the System Relocation Pointer (SRP), for pointing to the physical address of the sector to which valid system data is to be relocated, the SRP always being in a different block of sectors to the Write Pointer (WP) and the System Write Pointer
30 (SWP).
20

35 31. A memory system according to any of claims 28 to 30, wherein the controller is configured so that if the COB contains one of said write pointers (WP, RP, SWP, SRP) at the
40 time when the controller needs to erase a said COB because obsolete data has just been created in another block, the controller proceeds with creating a new COB but postpones the erasure of the old COB, herein referred to as the Pending Obsolete Block (POB), until all erased sectors in the POB have
45 been filled and said Pointer moves on to the next erased block to be used, as defined by the controller, at which time any valid (not obsolete) data in the POB is relocated by the
50 controller and the POB is erased.
55

5 32. A memory system according to claim 29, wherein the
controller is configured to store in a temporary memory of the
memory system respective lists of logical sector addresses
10 corresponding to sectors in the memory to which relocated data
5 has been written to by the RP (herein referred to as the
Relocation Sector List or RSL), the SWP (herein referred to as
the Write System Sector List or WSSL), and the SRP (herein
15 referred to as the System Relocation Sector List or SRSL)
since the SAT was last updated, and the controller is
10 configured to store in said temporary memory corresponding
lists of the order of blocks which have been used by the RP,
20 SWP and SRP (herein referred to as the Relocation Block List
(RBL), the Write System Block List (WSBL) and the System
Relocation Block List (SRBL)).

15 33. A memory system according to any preceding claim, wherein
25 in addition to writing data structures to the memory from the
host processor, the controller also generates and writes to
the memory data designated as control information, and the
30 controller is configured so as to write such control
information in one or more different ones (Control Blocks or
CBs) of the blocks of memory sectors to those in which data
35 structures received from the host processor are written.

25 34. A memory system according to claim 33, wherein the
controller is configured to store in at least one said Control
40 Block a list of the block addresses of all the SAT blocks.

35 35. A memory system according to claim 33 or claim 34, wherein
45 the controller is configured to store the block addresses of
said one or more Control Blocks in a dedicated block (the Boot
Block or BB) of the memory, this dedicated block being the
first block of sectors in the memory which does not contain
50 any bad sectors.

5 36. A memory system according to claim 34, as dependent from
claim 7, 8 or 9, wherein said list of all the SAT block
addresses is in the form of a plurality of list portions
10 (Table Block Lists or TBLs), and each said portion contains
5 the block addresses of a group of logically contiguous SAT
blocks and any ASBs corresponding thereto.

15 37. A memory system according to any of claims 7 to 9, wherein
the controller is configured to store in a temporary memory of
10 the memory system a list (the Write Sector List or WSL) of
logical sector addresses for data structures which have been
20 written by the Write Pointer (WP) since the SAT was last
updated.

25 38. A memory system according to claim 37, wherein the
controller is configured to also store in said temporary
memory the order in which blocks have been used by the Write
Pointer (WP) for writing data since the last update of the
30 SAT, this order being stored in the form of a list (the Write
20 Block List or WBL) of block addresses of the blocks in which
the updated sectors whose addresses are held in the WSL are
located.

35 39. A memory system according to claim 38, wherein the WSL has
25 a predetermined size and once the WSL is full at least one SAT
40 block or ASB block is updated and the WSL and WBL are emptied.

45 40. A memory system according to claim 38, wherein the
controller stores a starting physical sector address, and the
30 links between blocks containing sectors to which data has been
written by the controller since the last update of a SAT or
ASB block, in a said Control Block of the solid state memory.

50

55

5 41. A memory system according to any preceding claim, wherein each said sector consists of a single "page" of memory, namely one row of memory cells in a said block of memory sectors.

10 42. A memory system according to any preceding claim, wherein the controller is configured to write data to, and read data from, the memory sectors in uniformly sized data segments.

15 43. A memory system according to claim 42, wherein all the 10 memory sectors are the same size and each said data segment is equal in size to the size of a said memory sector.

20 44. A memory system according to any preceding claim, further including a temporary cache memory in which the controller is 25 configured to store a group of contiguous SAT entries including the SAT entry most recently accessed from the SAT by the controller.

30 45. A memory system according to claim 43, when dependent from 20 claim 8, wherein the controller is configured to create in said temporary cache memory a list (ASBL) of physical 35 addresses of all ASBs and the SAT blocks with which they are associated which is updated each time a SAT sector write operation is performed.

25 46. A memory system according to any preceding claim, wherein 40 the solid state memory comprises a single memory array in the form of a single memory chip.

45 47. A memory system according to any of claims 1 to 45, wherein the solid state memory comprises a memory array formed 50 by a plurality of memory chips.

55 48. A memory system according to any of claims 1 to 45, wherein the solid state memory comprises a plurality of memory

5 arrays in the form of a plurality of memory chips, and wherein
the controller is configured to form the memory sectors in the
plurality of memory chips into a multiplicity of virtual
10 blocks, each said virtual block comprising one erasable block
of memory sectors from each said memory chip, and to sort said
virtual blocks into ones which are treated as erased and ones
which are treated as not erased.

15 49. A memory system according to claim 48, wherein the
controller is configured to compile a list of the virtual
blocks treated as erased and store this in temporary memory in
20 the memory system, and to control the Write Pointer (WP) to
move from one chip to another for each consecutive sector
write operation, starting at one sector in one erasable block
15 of the virtual block and moving consecutively to one sector in
each of the other erasable blocks in the virtual block until
one sector has been written in each erasable block of the
virtual block, and then moving back to the chip in which the
30 first sector was written and proceeding in a similar manner to
fill another one sector in each erasable block of the virtual
block, and so on until the virtual block is full of data, and
then to move the Write Pointer (WP) on to the next virtual
35 block in said list of virtual blocks being treated as erased,
and fill this next virtual block in a similar manner.

25 50. A memory system according to claim 49, wherein the
controller is configured so that for every n contiguous sector
write operations the controller executes for a multiple sector
write command received from the host processor, where n is
40 less than or equal to the number of solid state memory chips
in the memory system, the controller writes substantially
concurrently to one sector in each of n of the chips.

50 51. A memory system according to claim 49 or claim 50, wherein
35 the controller is configured to carry out erasure of any said

5 virtual block by concurrently erasing all the erasable blocks
in the virtual block.

10 52. A memory system for connection to a host processor, the
5 memory system comprising:

15 a solid state memory comprising a plurality of solid state
memory chips each having non-volatile memory sectors which are
individually addressable and which are arranged in erasable
10 blocks of sectors, each said sector having a physical address
defining its physical position in the memory;
and a controller for writing data structures to and reading
20 data structures from the memory, wherein:
the controller forms the erasable blocks into virtual blocks,
each said virtual block comprising an erasable block from each
25 of the memory chips, and the controller sorts the virtual
blocks into ones which are treated as erased and ones which
are treated as not erased, and the controller fills one
virtual block with data prior to moving on to the next virtual
30 block to be filled, and each virtual block is filled by
writing to the memory sectors thereof in a repeating sequence
in which the controller writes to one memory sector in each of
the erasable blocks of the virtual block one after another
35 whereby consecutively written sectors are in different chips.

25 53. A memory system according to claim 52, wherein the
controller is configured so that for every n contiguous sector
40 write operations the controller executes for a multiple sector
write command from the host processor, where n is less than or
equal to the number of solid state memory chips in the memory
45 30 system, the controller writes substantially concurrently to
one sector in each of n of the chips.

50 54. A controller for writing data structures to and reading
data structures from a solid state memory having non-volatile
35 memory sectors which are individually addressable and which

5 are arranged in erasable blocks of sectors, each said sector having a physical address defining its physical position in the memory, wherein the controller includes:

10 means for translating logical addresses received from a host processor of a memory system in which the controller is used to physical addresses of said memory sectors in the memory, and for sorting the blocks of sectors into blocks which are treated as erased and blocks which are treated as not erased;

15 and a Write Pointer (WP) for pointing to the physical address of a sector which is to be written to from the host processor, said Write Pointer (WP) being controlled by the controller to move in a predetermined order through the physical addresses of the memory sectors in any block which is treated as erased and, when the block has been filled, to move to another of the

20 15 erased blocks, and wherein the controller is configured so that, when a sector write command is received by the controller from the host processor, the controller translates a logical sector address received from the host processor to a physical address to which data is written by allocating for

25 30 said logical address that physical address to which said Write Pointer (WP) is currently pointing; and wherein the controller is configured to compile a table (the SAT) of logical addresses with respective physical addresses which have been allocated therefor by the

35 25 controller, and to update the SAT less frequently than memory sectors are written to with data from the host processor.

40

55. A method of controlling reading and writing of data structures to and from a solid state memory having non-

45 30 volatile memory sectors which are individually addressable and which are arranged in erasable blocks of sectors, each said sector having a physical address defining its physical position in the memory, the method comprising the steps of:

50 sorting the blocks of sectors into blocks which are treated as 35 erased and blocks which are treated as not erased;

55

5 providing a Write Pointer (WP) for pointing to the physical
address of a sector which is to be written to, and controlling
said at least one Write Pointer (WP) so as to move in a
10 predetermined order through the physical addresses of the
5 memory sectors of any block which is treated as erased, and
when the block has been filled to move to another of the
erased blocks and, when a sector write command is received
15 from the host processor, translating a logical address
received from the host processor to a physical address to
10 which data is written by allocating for said logical address
that physical address to which said Write Pointer (WP) is
20 currently pointing;
storing in non-volatile solid state memory a table (the SAT)
of logical addresses with respective physical addresses which
25 have been allocated therefor by the controller;
and updating the SAT less frequently than memory sectors are
written to with data from the host processor.

30 56. A memory system for connection to a host processor, the
20 system comprising:

a solid state memory having non-volatile memory sectors which
are individually addressable and which are arranged in
35 erasable blocks of sectors, each said sector having a physical
address defining its physical position in the memory;
25 and a controller for writing data structures to and reading
data structures from the memory, wherein the controller
40 includes means for translating logical addresses received from
the host processor to physical addresses of said memory
sectors in the memory; and wherein
45 each said memory sector (1) is physically partitioned into a
data area (300) and a spare area (302) and the controller is
configured so as to write overhead data (OD) comprising header
data and error correction code data (ECC) at a position in the
50 sector which is offset from the start of the data area (300)
35 of the sector and to write user data, received from the host

5 processor, in the space remaining in the sector, on either
side of the overhead data (OD).

10 57. A memory system according to claim 56, wherein said
5 overhead data (OD) is offset by an amount which is determined
by at least one bit of the user data to be written to the
sector.

15

20

25

30

35

40

45

50

55

1/21

Fig.1.

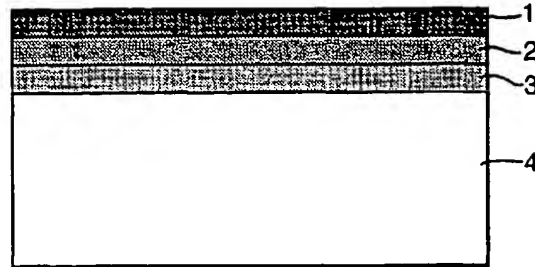


Fig.2.

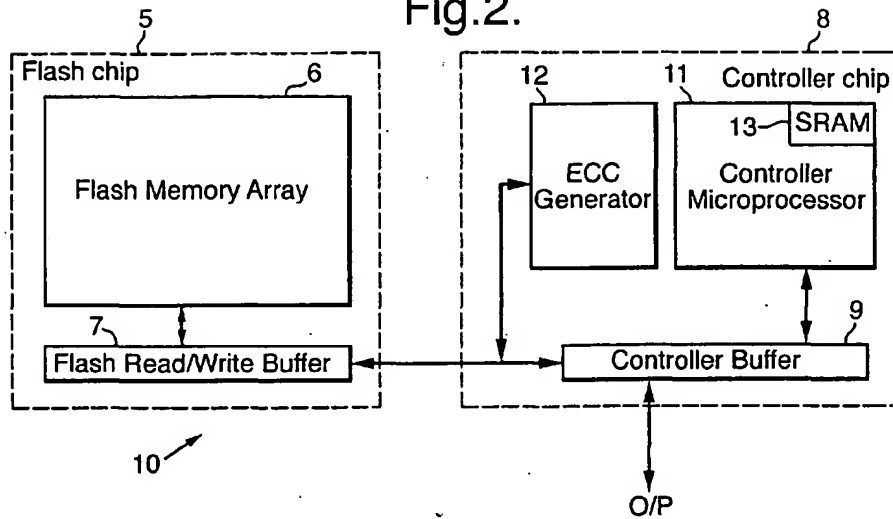


Fig.3.

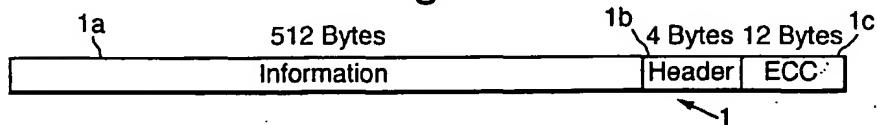


Fig.4.

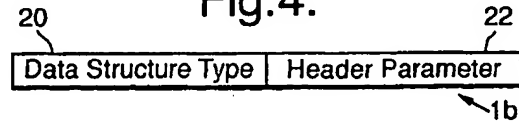


Fig.5.

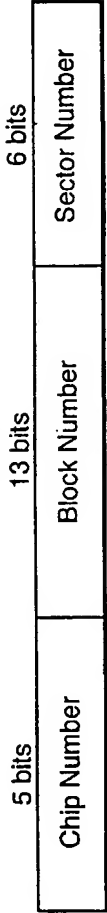


Fig.6.

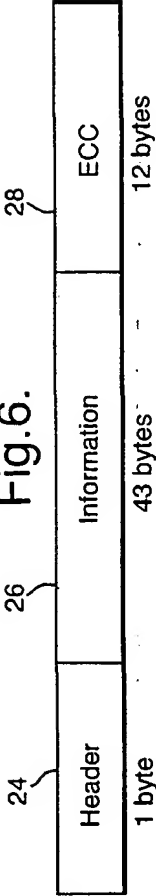


Fig.7.

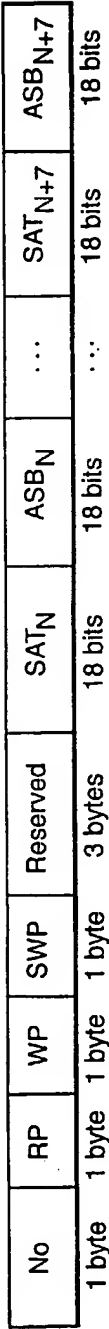


Fig.8.



Fig.9.

SAT Block Index	SAT Block Address	ASB Address	LWP	NVP	ASB Page 0	...	ASB Page N
2 bytes	3 bytes	3 bytes	1 byte	1 byte	1 byte	...	1 byte

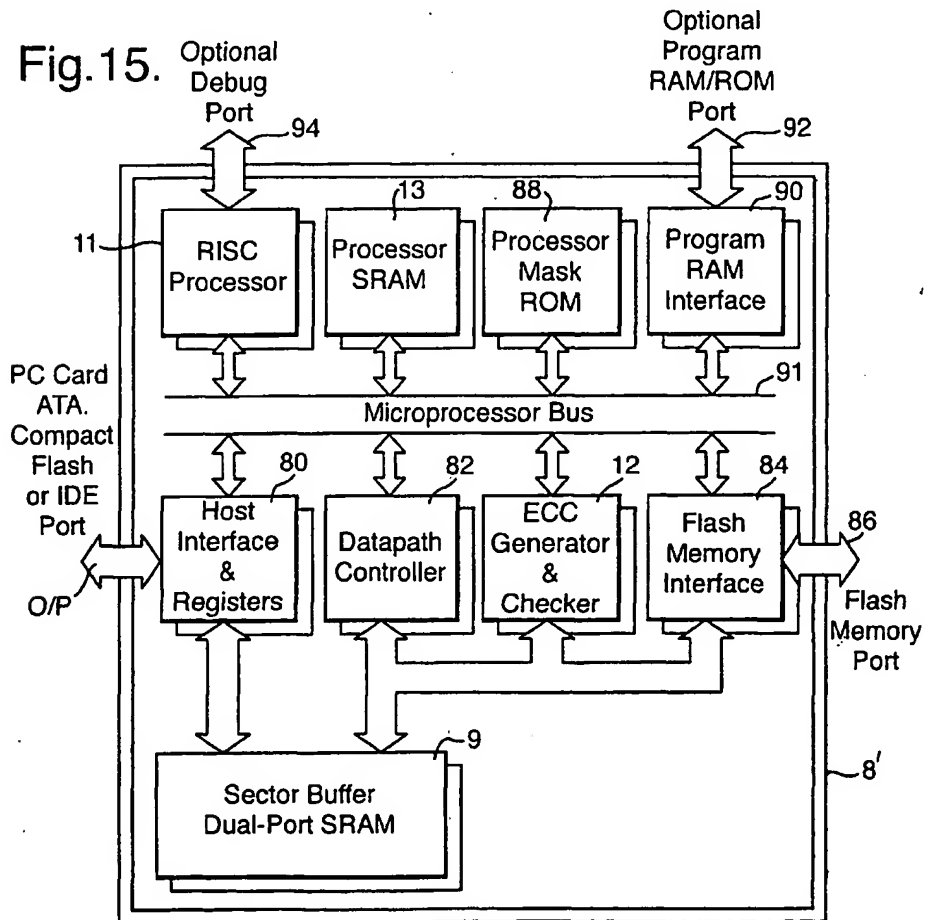
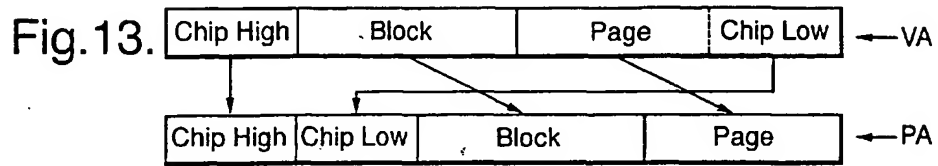
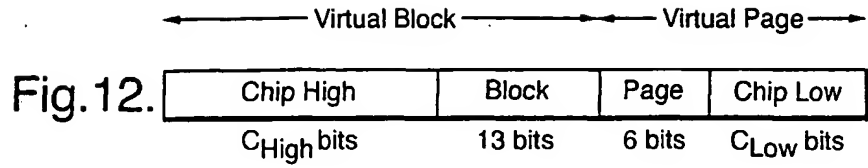
Fig.10.

28	Block Number	Obsolete or Deleted Sector Mask
	4 bytes	32 bytes

Fig.11.

Chip 0 Block 0	Chip 1 Block 0	Chip 2 Block 0	Chip 3 Block 0
0	1	2	3
4	5	6	Etc.

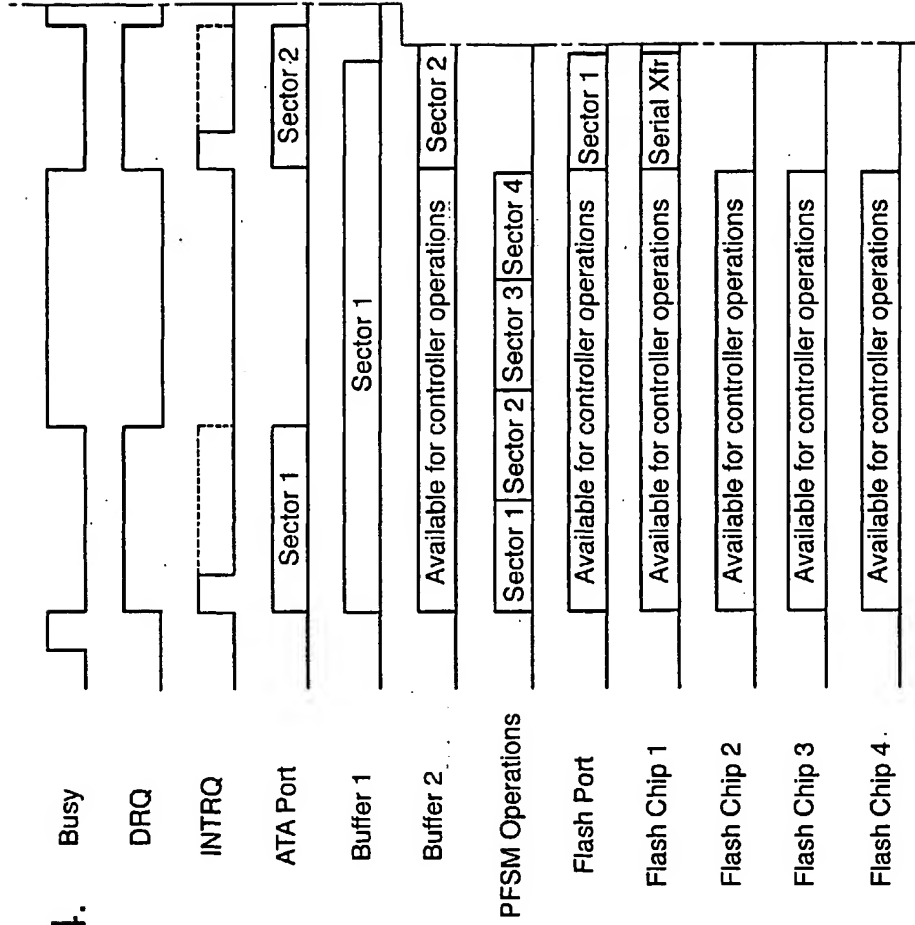
4/21



SUBSTITUTE SHEET (RULE 26)

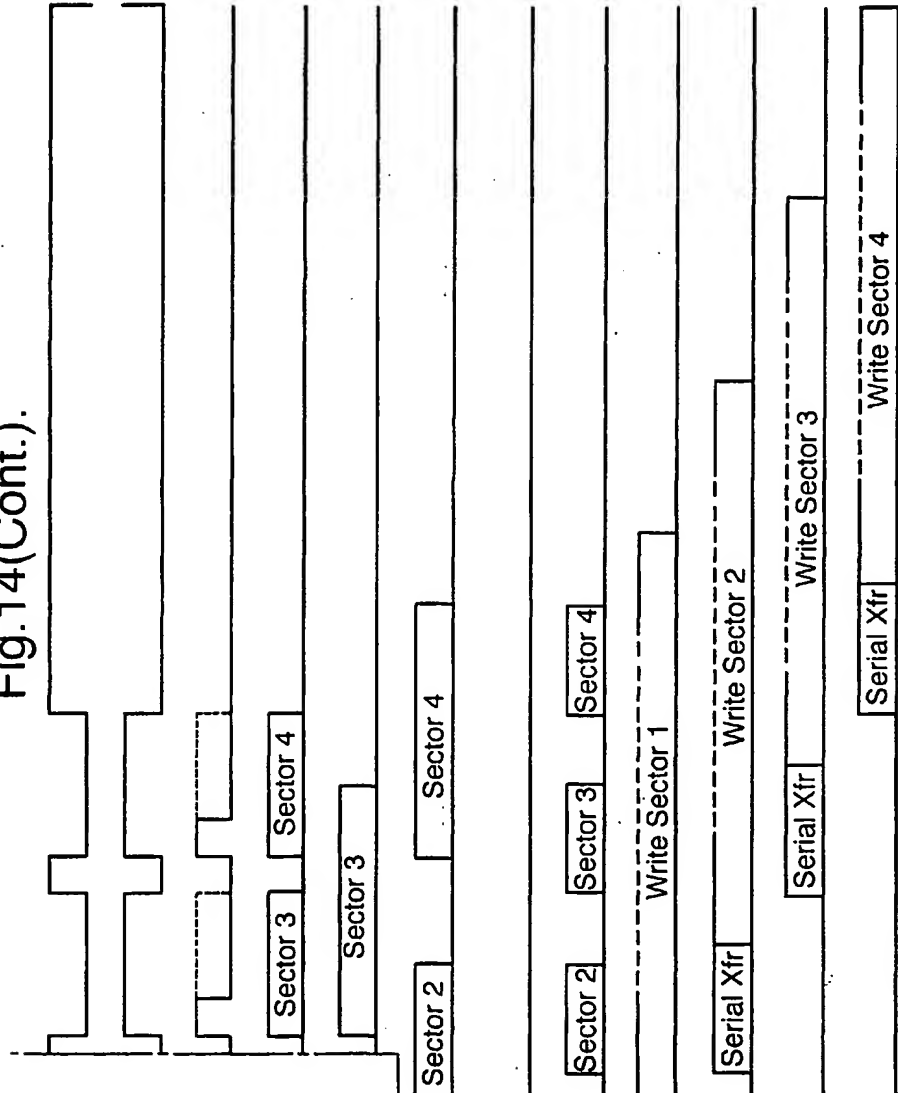
5/21

Fig. 14.



6/21

Fig. 14(Cont.).



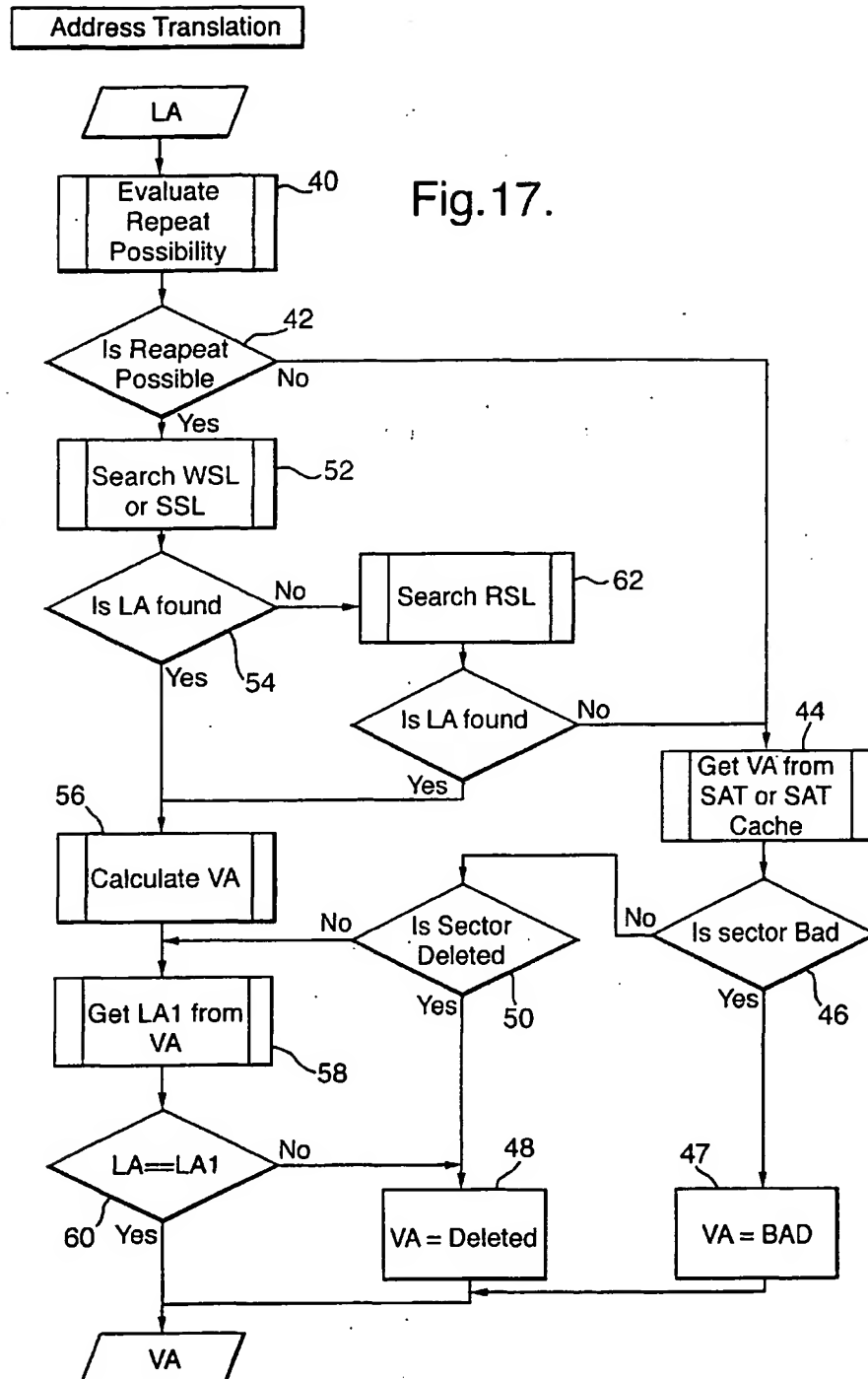
7/21

Capacity Allocation Table

	8MB Card		64MB Card		512MB Card	
	Number of Blocks	% of Capacity	Number of Blocks	% of Capacity	Number of Blocks	% of Capacity
Total Capacity	1024	100%	8192	100%	65536	100%
Sectors	1007	98.1%	8132	99.2%	65131	99.4%
Boot Block	1	0.1%	1	0.01%	1	0.00%
Control Block	3	0.3%	4	0.05%	4	0.00%
Sector Address Table	6	0.6%	48	0.6%	384	0.6%
Additional SAT Blocks	6	0.6%	8	0.1%	8	0.01%
Obsolete Sectors	1	0.1%	1	0.01%	1	0.00%
Erased Buffer	2	0.2%	2	0.02%	2	0.00%
Spare Blocks						

Fig.16.

8/21



9/21

Fig.18.

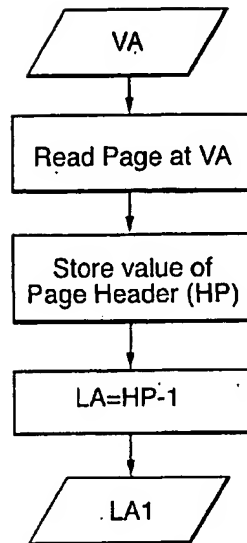
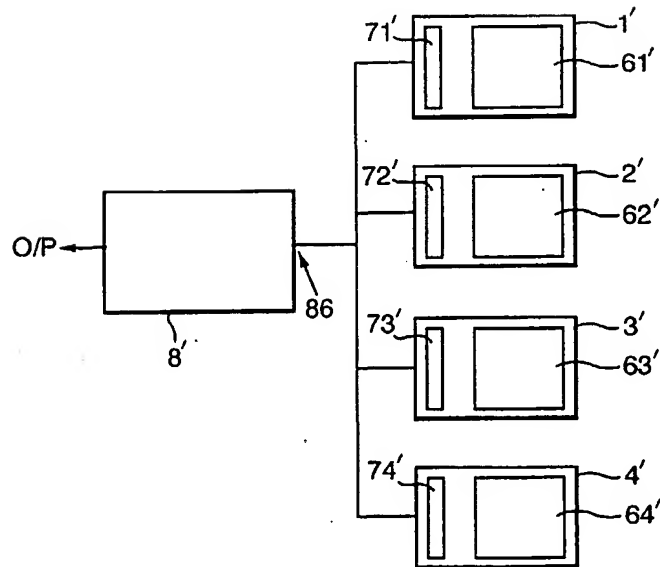


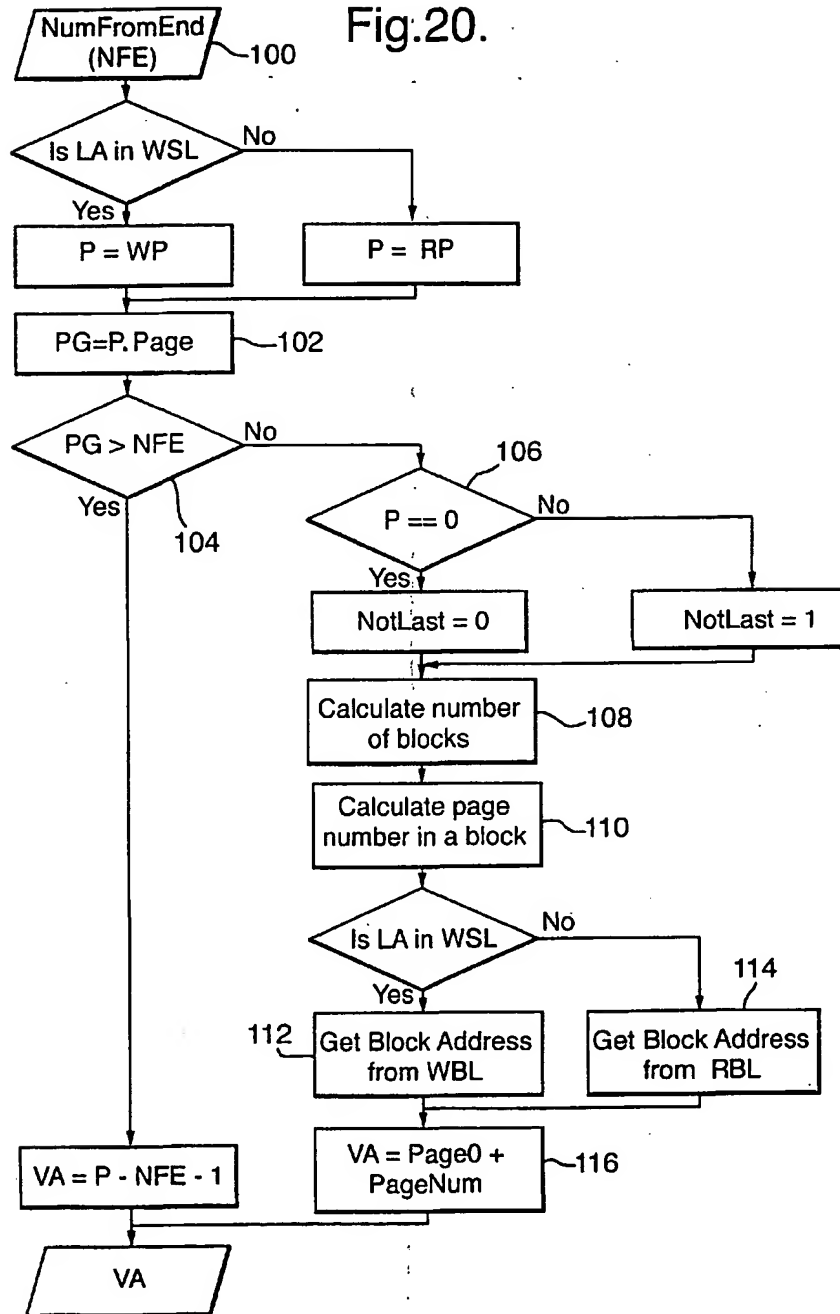
Fig.19.



SUBSTITUTE SHEET (RULE 26)

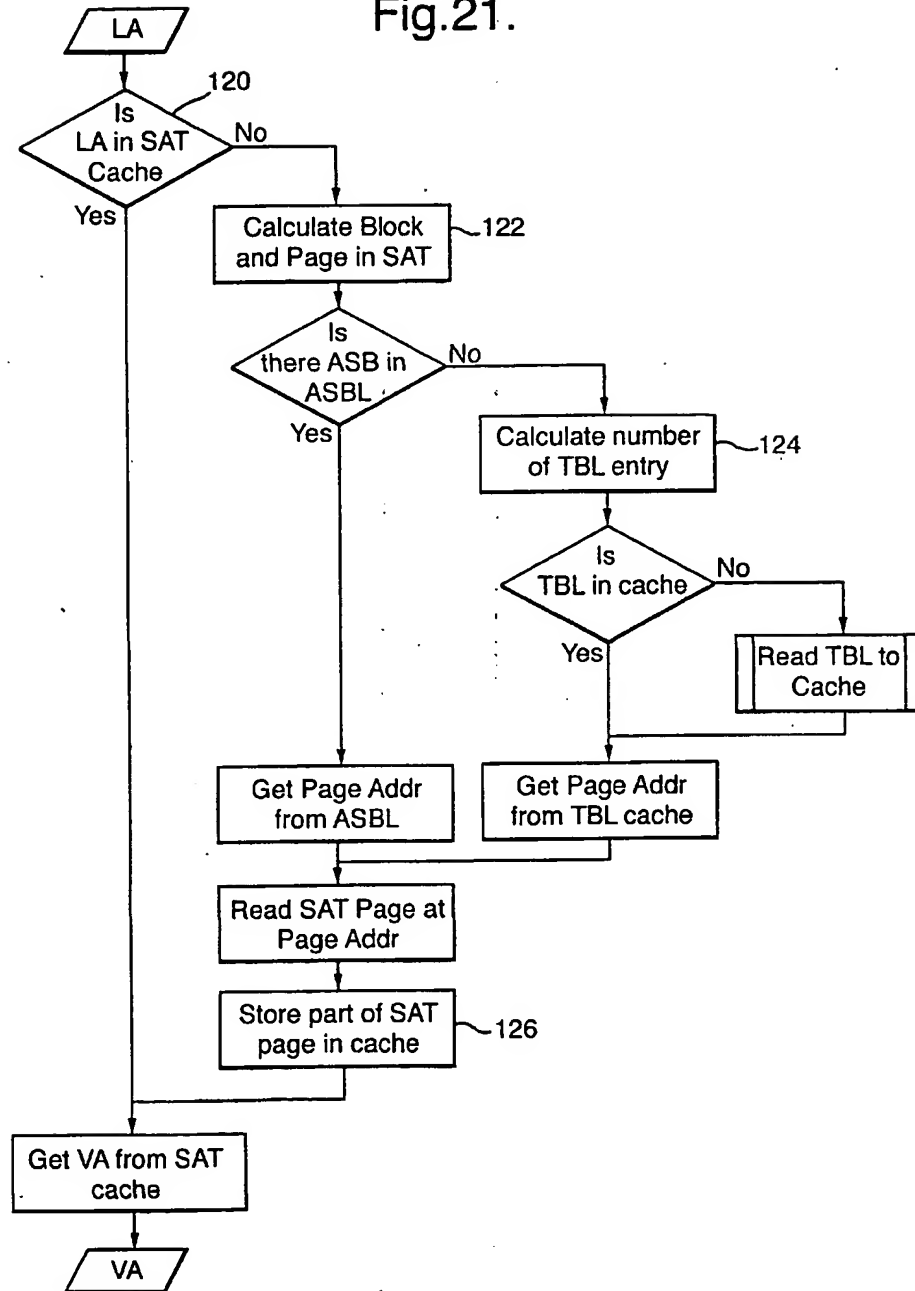
10/21

Fig.20.



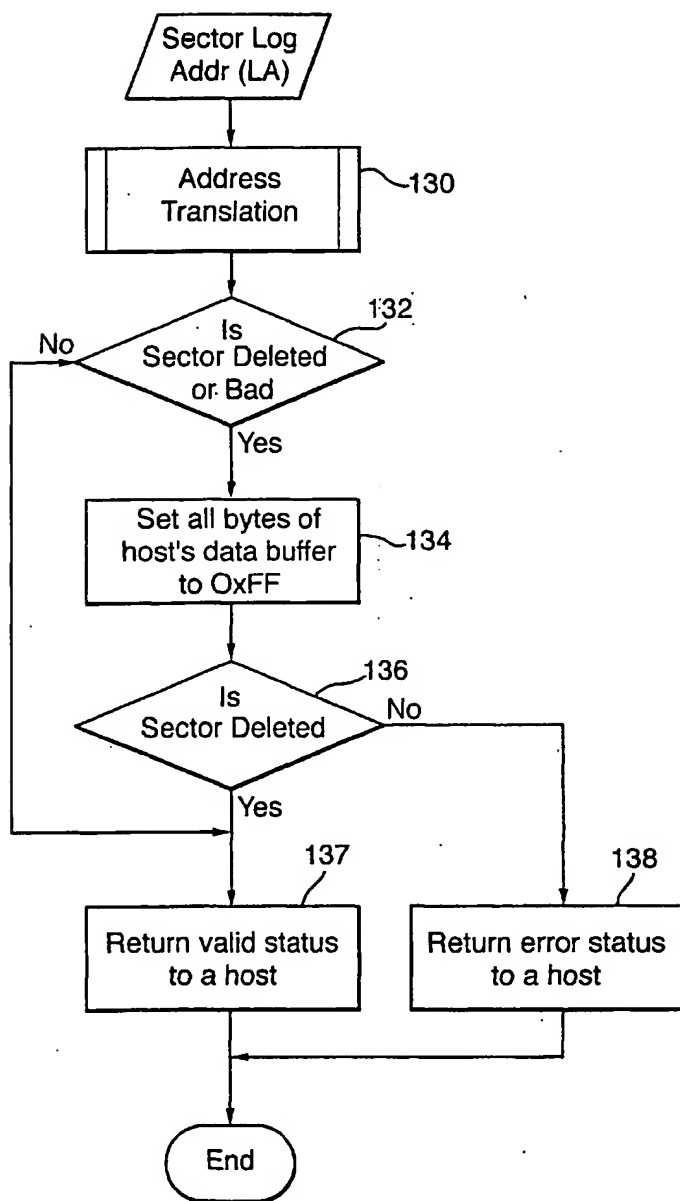
11/21

Fig.21.



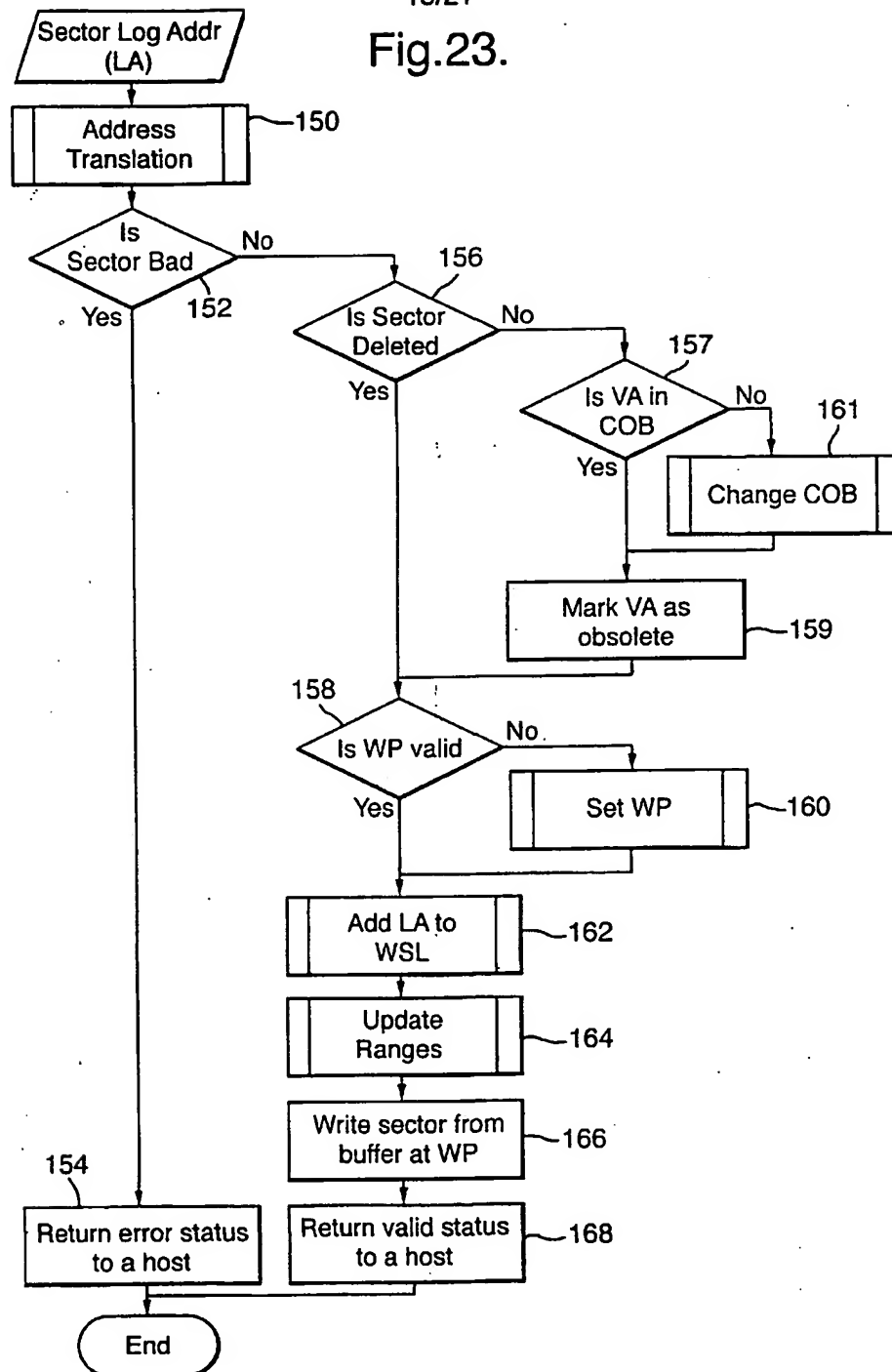
12/21

Fig.22.



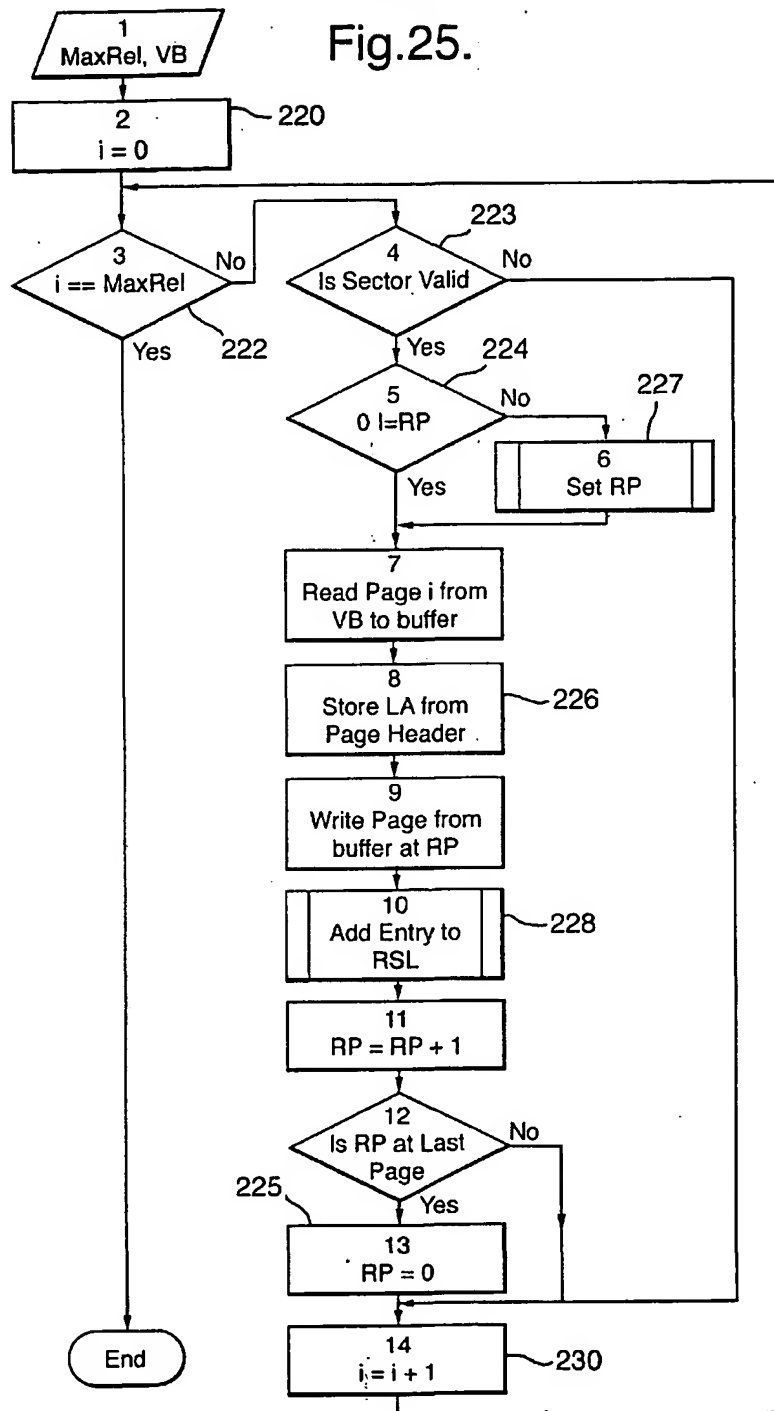
13/21

Fig.23.



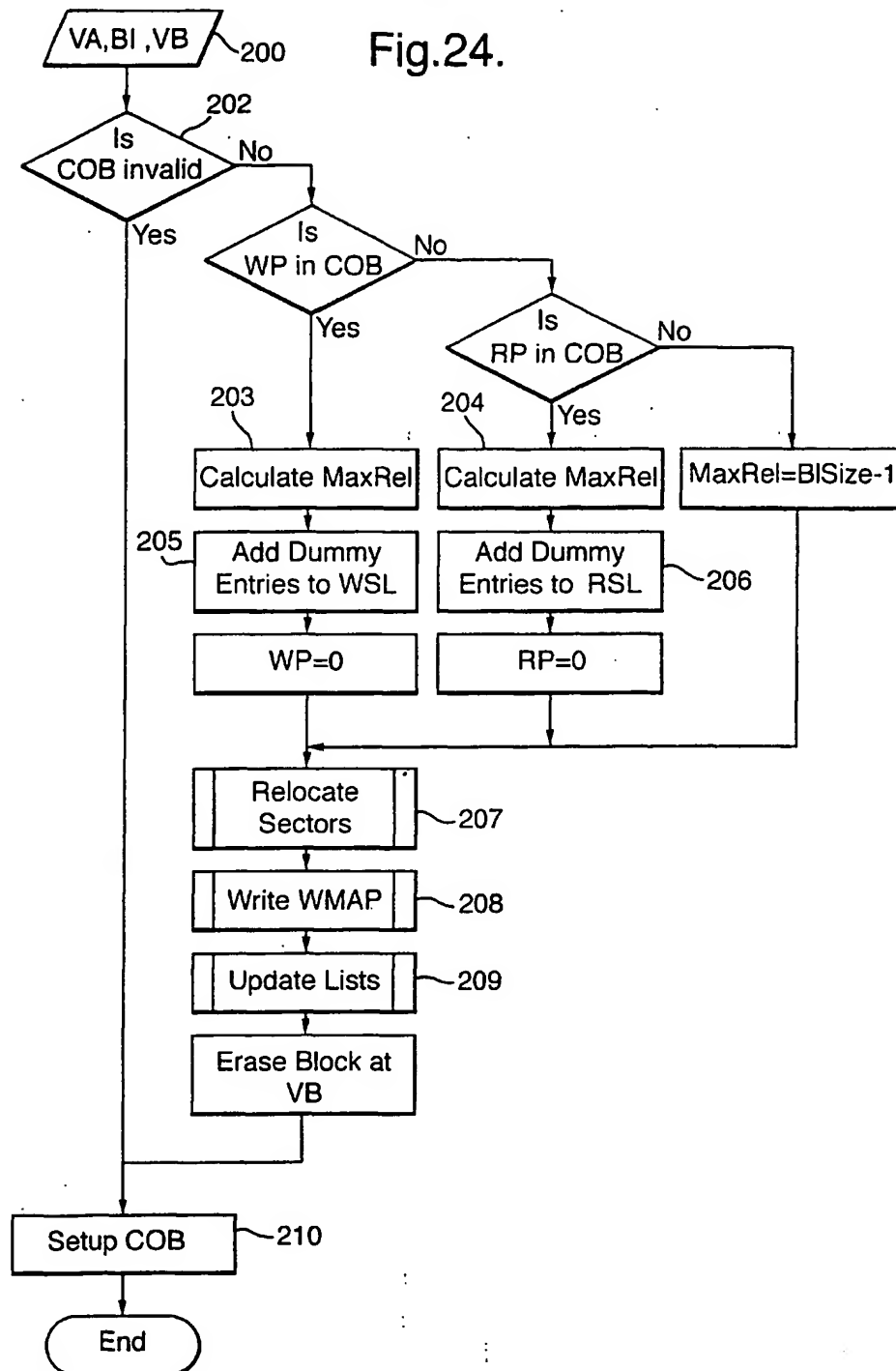
15/21

Fig.25.



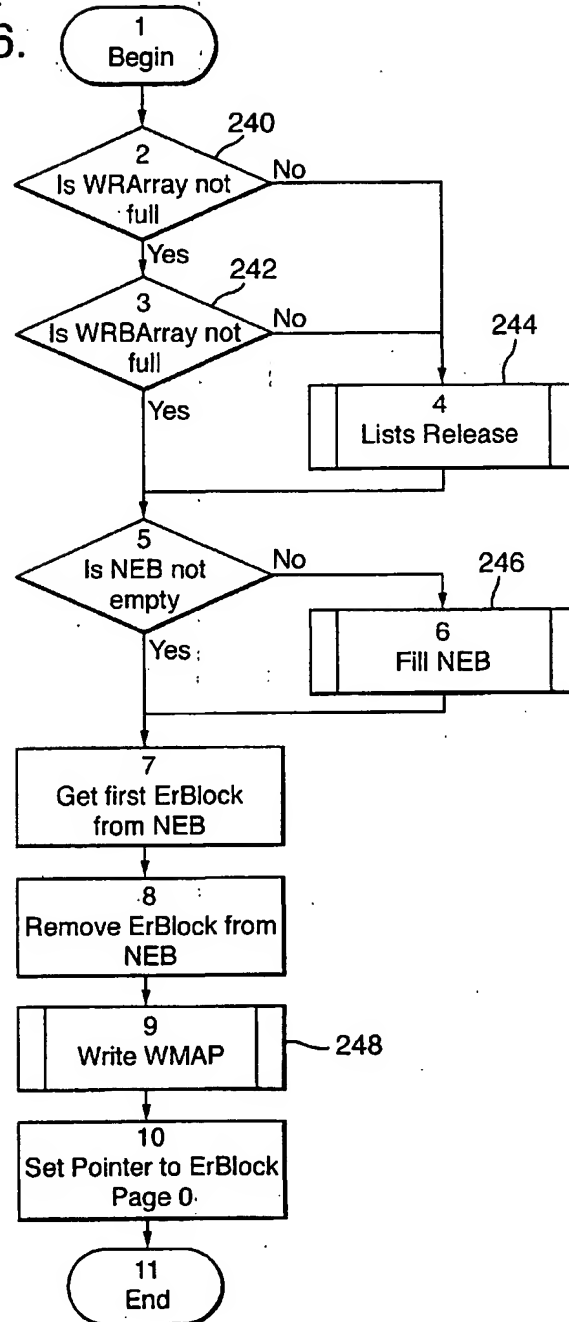
14/21

Fig.24.



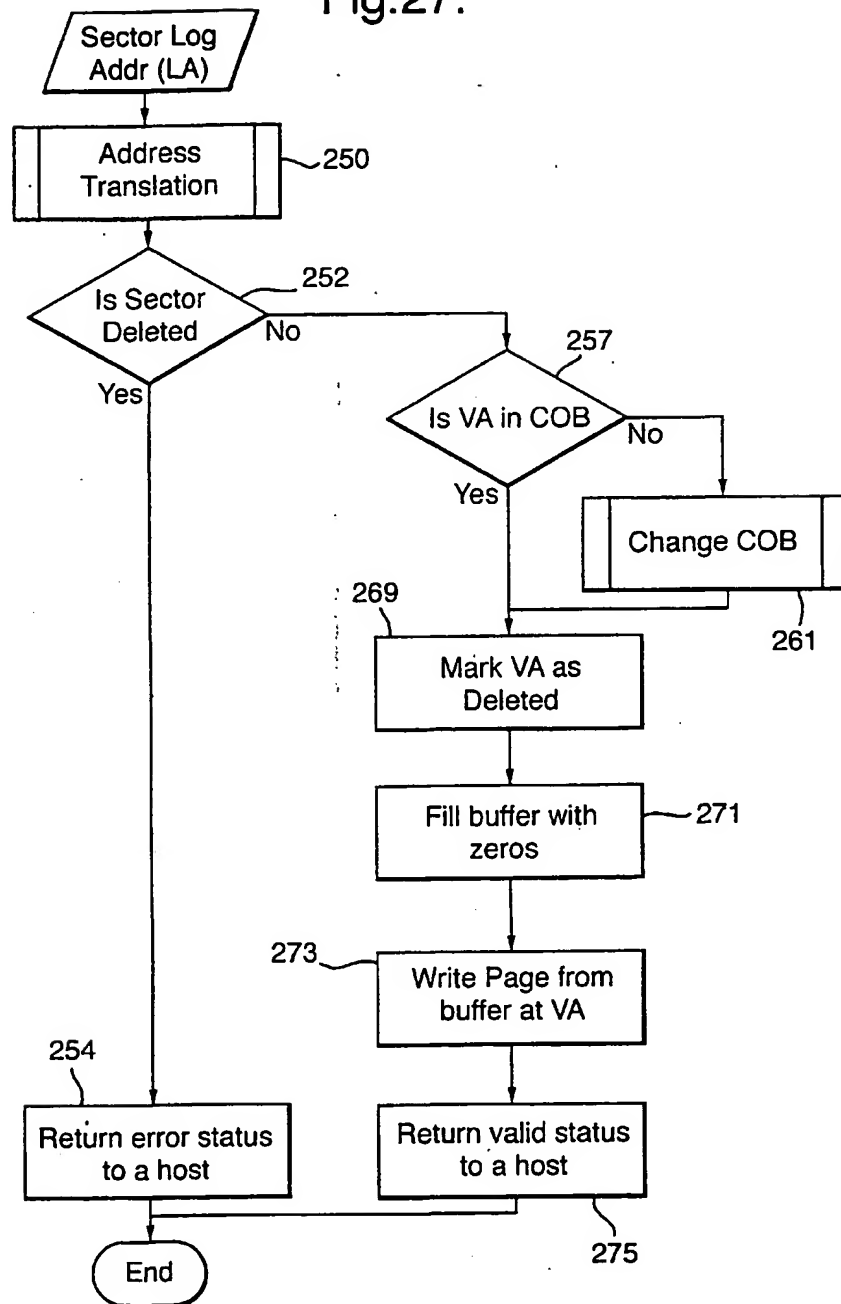
16/21

Fig.26.



17/21

Fig.27.



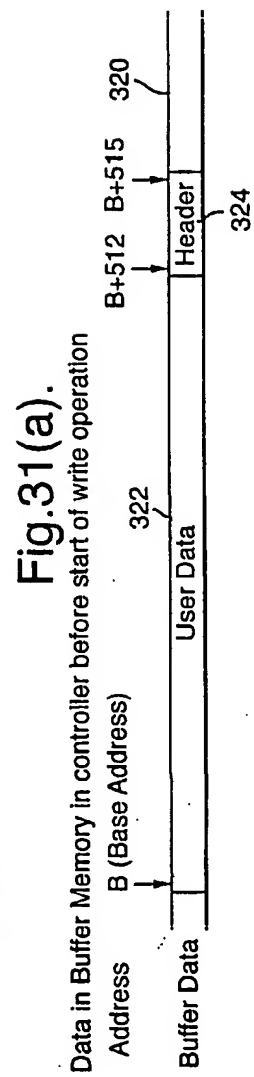
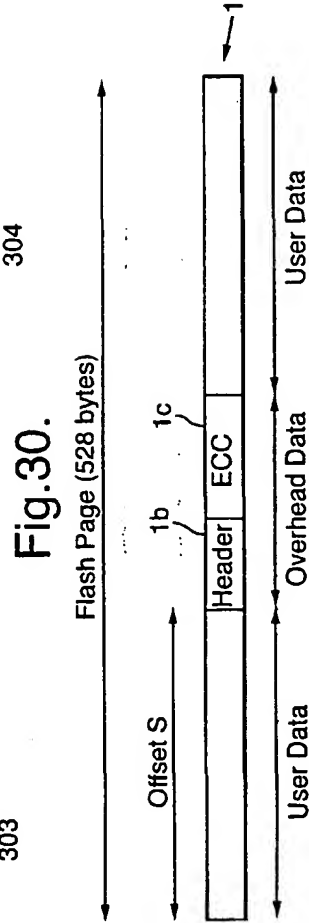
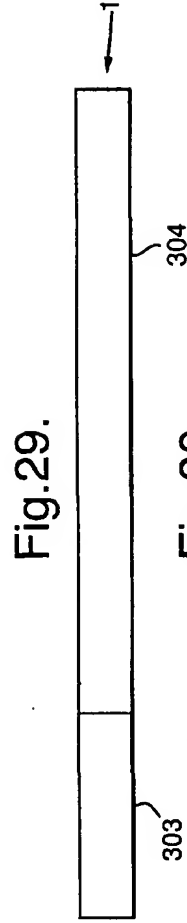
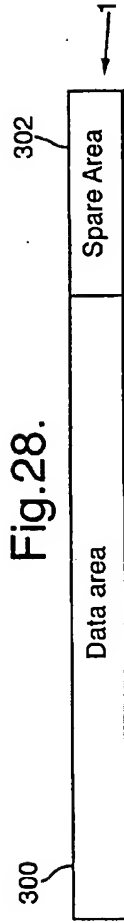


Fig.31(b).

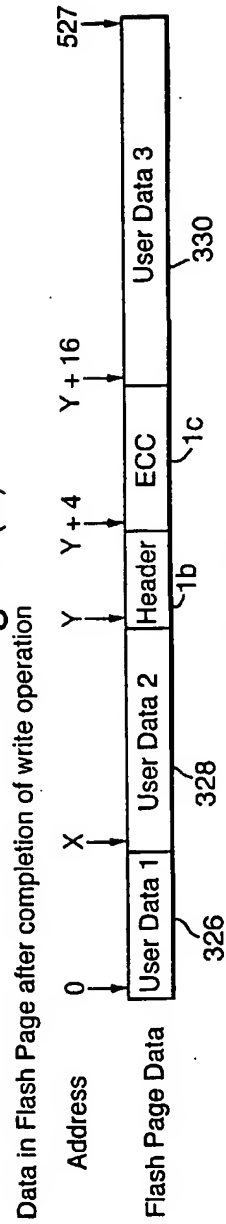


Fig.33.

Data in Buffer Memory in controller after completion of read operation

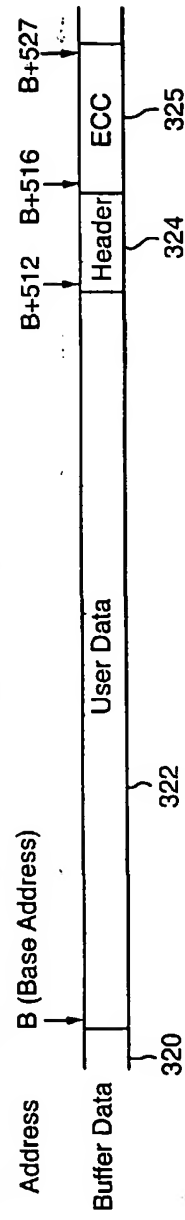
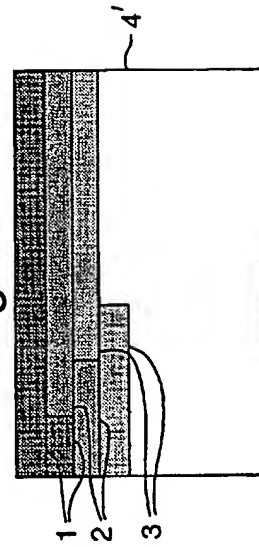


Fig.35.



20/21

Fig.32.
Sequence of commands to controller data transfer hardware

Command Sequence	Command Parameters			Command Description		
	Flash Start Address	Buffer Start Address	No. of Data Xfr Cycles	ECC Generator Mode	Flash Clock	Summary
Cmnd 1	n/a	B	516	Generator on Output register disabled	off	Generate ECC for 512 bytes of User Data + Header
Cmnd 2	0	B	Y	Generator off Output register disabled	on	Transfer User Data 1 + User Data 2 to Flash buffer
Cmnd 3	continue	B + 512	4	Generator off Output register disabled	on	Transfer Header to Flash buffer
Cmnd 4	continue	n/a	12	Generator off Output register enabled	on	Transfer ECC to Flash buffer
Cmnd 5	continue	B + Y	512 - Y	Generator off Output register disabled	on	Transfer User Data 3 to Flash buffer Write Flash buffer to Flash array

Fig.34.

Sequence of commands to controller data transfer hardware

Command Sequence	Command Parameters			Command Description		
	Flash Start Address	Buffer Start Address	No. of Data Xfr Cycles	ECC Generator Mode	Flash Clock	Summary
Cmnd 1	0	B	X	Generator on Output register disabled	on	Read Flash page to Flash buffer Transfer User Data 1 from Flash buffer
Cmnd 2	continue	B + X	Y - X	Generator on Output register disabled	on	Transfer User Data 2 from Flash buffer
Cmnd 3	continue	B + 512	16	Generator off Output register disabled	on	Transfer Header + ECC from Flash buffer
Cmnd 4	continue	B + Y	512 - Y	Generator on Output register disabled	on	Transfer User Data 3 to Flash buffer
Cmnd 5	n/a	B + 512	16	Generator on Output register disabled	off	Transfer Header + ECC from Buffer to ECC Generator

INTERNATIONAL SEARCH REPORT

International Application No
PCT/GB 00/00550

A. CLASSIFICATION OF SUBJECT MATTER

IPC 7 G06F3/06

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the International search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	GB 2 291 991 A (MEMORY CORP PLC) 7 February 1996 (1996-02-07) page 5, line 15 -page 8, line 5; figures	1,2,4, 10,11, 15,17, 20,46, 47,52, 54-56
A	WO 97 37296 A (SINCLAIR ALAN WELSH ;MEMORY CORP PLC (GB)) 9 October 1997 (1997-10-09) page 3, line 21 -page 7, line 6; figures -/-	1,52, 54-56

☒ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"A" document member of the same patent family

Date of the actual completion of the international search

25 May 2000

Date of mailing of the international search report

02/06/2000

Name and mailing address of the ISA

European Patent Office, P.B. 5818 Postfach 2
NL - 2280 HV Rijswijk
Tel. (+31-70) 340-2040, Tx. 31 651 epo nl,
Fax: (+31-70) 340-3016

Authorized officer

Moens, R

INTERNATIONAL SEARCH REPORT

International Application No
PCT/GB 00/00550

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT		
Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	EP 0 712 067 A (MITSUBISHI ELECTRIC CORP) 15 May 1996 (1996-05-15) column 4, line 37 -column 6, line 32; figures	1-4, 10, 15, 17, 24, 46, 47, 52, 54-56
A	EP 0 522 780 A (IBM) 13 January 1993 (1993-01-13) claims 1, 9, 13	1, 52, 54-56
A	WO 94 20906 A (SYSTEMS LTD M ;SYSTEMS INC M (US)) 15 September 1994 (1994-09-15) page 2, line 21 -page 5, line 2	1, 48, 49, 52
A	US 5 602 987 A (HARARI ELIYAHOU ET AL) 11 February 1997 (1997-02-11) column 8, line 39 - line 67; figures 1A, 2	1, 13, 14, 16, 56, 57

1

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No

PCT/GB 00/00550

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
GB 2291991 A	07-02-1996	CN 1198226 A	04-11-1998
		EP 0852766 A	15-07-1998
		WO 9712325 A	03-04-1997
		JP 11511879 T	12-10-1999
WO 9737296 A	09-10-1997	EP 0896699 A	17-02-1999
EP 0712067 A	15-05-1996	JP 8137634 A	31-05-1996
		US 5905993 A	18-05-1999
EP 0522780 A	13-01-1993	JP 2582487 B	19-02-1997
		JP 5027924 A	05-02-1993
		DE 69223287 D	08-01-1998
		DE 69223287 T	28-05-1998
		US 5524230 A	04-06-1996
WO 9420906 A	15-09-1994	US 5404485 A	04-04-1995
		AU 6269994 A	26-09-1994
		CN 1098526 A	08-02-1995
		DE 69414556 D	17-12-1998
		DE 69414556 T	06-05-1999
		EP 0688450 A	27-12-1995
		FI 954235 A	08-11-1995
		IL 108766 A	05-12-1996
		JP 8510072 T	22-10-1996
		ZA 9401446 A	26-09-1994
US 5602987 A	11-02-1997	US 5297148 A	22-03-1994
		US 5535328 A	09-07-1996
		US 5991517 A	23-11-1999
		DE 69024086 D	25-01-1996
		DE 69024086 T	20-06-1996
		DE 69033262 D	30-09-1999
		DE 69033262 T	24-02-2000
		DE 69033438 D	02-03-2000
		EP 0392895 A	17-10-1990
		EP 0617363 A	28-09-1994
		EP 0618535 A	05-10-1994
		EP 0675502 A	04-10-1995
		EP 0935255 A	11-08-1999
		JP 2292798 A	04-12-1990
		US 5671229 A	23-09-1997
		US 5719808 A	17-02-1998
		US 5936971 A	10-08-1999
		US 5418752 A	23-05-1995
		US 5877986 A	02-03-1999
		US 5862080 A	19-01-1999
		US 5999446 A	07-12-1999